AFRL-IF-RS-TR-2004-33
Final Technical Report
February 2004

MULTILEVEL COORDINATION MECHANISMS FOR REAL-TIME AUTONOMOUS AGENTS

University of Michigan

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J356

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-33 has been reviewed and is approved for publication

APPROVED: /s/

FRANK H. BORN
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2004	Final Mar 98 – Apr 03

4. TITLE AND SUBTITLE
MULTILEVEL COORDINATION MECHANISMS FOR REAL-TIME AUTONOMOUS AGENTS

5. FUNDING NUMBERS
C - F30602-98-2-0142
PE - 63760E
PR - AGEN
TA - T0
WU - 06

6. AUTHOR(S)
Edmund H. Durfee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Michigan
3003 South State Street
Ann Arbor Michigan 48109-1274

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-33

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Frank H. Born/IFTB/(315) 330-4726/ Frank.Born@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This report summarizes the research and development efforts performed in the DARPA-sponsored project in Multilevel Coordination Mechanisms for Real-Time Autonomous Agents, as part of the DARPA Control of Agent Based Systems (CoABS) program. The work was performed from May of 1998 to April of 2003. During the course of the project, significant advances have been made in the area of plan-based control and coordination of semi-autonomous agents. Specifically, techniques for modeling and coordinating agents based on hierarchical representations of their plans have been shown to be efficient and effective in enabling multiple agents, each with its own sophisticated plans and objectives, to predict and resolve unintended conflicts between their operations, as well as to anticipate and exploit opportunities for cooperation. The basic research results have been implemented into a Multilevel Coordination Agent (MCA) that operates on the CoABS Grid. The MCA has been demonstrated as part of the Coalition Agents Experiment (CoAX), culminating in the October 2002 demonstration at the Naval Warfare Development Center in Newport, RI.

14. SUBJECT TERMS
Agents Systems, Multi-Agent, Software Agent, Multi-Level Coordination, Planning, Scheduling, Autonomous Agents

15. NUMBER OF PAGES
95

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1 Introduction

1.1 Project Objectives

Agile and rapid teaming, such as in the concerted application of coalition forces in uncertain, confrontational settings, requires efficiently coordinating mission plans that also give team members some room to improvise mission details around unexpected or previously unobservable events. The goal of this project has been to develop technologies that enable multi-level coordination: participating agents should be able to decide on the level of plan detail at which to make coordination commitments, based on the current circumstances and their needs to balance predictability to each other with flexibility to react autonomously. By focusing only on details of interactions that matter in particular situations, multi-level coordination techniques will lead to scalable, efficient, and robust coordination outcomes. These benefits have been evaluated in analytical and empirical studies, and demonstrated through integrated experiments in simulated coalition operations tasks.

1.2 Project Approach

The principal insight behind this project's multi-level coordination technologies has been that hierarchical models of an agent's plans and goals can be exploited to support coordination that strikes a balance between predictability and flexibility that is more tailored for particular mission needs. Many problem domains can be effectively planned for in a hierarchical manner, where the broad outlines of behavior are laid out and then incrementally refined in time, space, and scope of participation. Rather than use hierarchy only as a means to an end (a detailed plan), multi-level coordination technologies allow agents to retain information at the various levels, and therefore *an agent can represent what it is doing at multiple levels of detail all at once*. Because it will need to engage in detailed coordination with some (physically or conceptually proximate) agents while at the same time coordinating in looser ways with other agents, *the agent can communicate models of itself at the right level of detail for different coordination relationships simultaneously*. By receiving such models from others, an agent can ensure that its decisions fit into the combined efforts of the agent system without getting bogged down in computations at unnecessarily detailed levels.

To represent a plan space, this project has adapted a representation of plans as nested procedures in increasing detail. Unlike traditional approaches where detailed plans are formulated in their entirety before execution, plans can be incrementally elaborated such that the most appropriate procedure to accomplish a particular goal is chosen only when that goal is to be achieved next. By comparing conditions that must hold over different intervals in agents' plans, timing relations that must hold for the plans to avoid unintentionally interfering with each other can be inferred. By propagating information about conditions that hold and about timing constraints between subplans upward through the hierarchy, relationships between plans at various levels of abstraction can thus be identified. This research includes analyses to ensure soundness and completeness of these methods, along with understanding their computational complexity. To amortize the computational costs, an additional innovation in this project is to enable agents to store previously computed coordination solutions for reuse under similar future circumstances.

The ability to detect and resolve unintended interactions at any of many levels of detail improves scalability, because an agent can use abstract models to quickly identify just those agents it needs to coordinate with, and can dynamically select the level of detail at which to coordinate with them. This research has included developing new, principled techniques for making these decisions, along with

recovery mechanisms for adapting to changing circumstances that mismatch previous coordination decisions.

Introducing the added dimension of being able to flexibly set the level of detail for agents to model each other at run time is becoming increasingly critical in the technologically and informationally-rich battlefield of the future. The methods that have been developed here should be widely applicable for systems in which the "right" level for coordination cannot be predetermined, but instead must be changeable in response to system needs and time-critical opportunities. The efficacy of our techniques has been demonstrated by implementing them as services in the Control of Agent Based Systems (CoABS) Grid, and used to coordinate the rapid deployment of (simulated) coalition forces where the lack of prior joint traing and shared understanding can otherwise lead to uncoordinated activities, with consequences ranging from minor (wasted effort) to major (friendly fire).

2 Research Results

This project has developed foundational underpinnings and operational prototypes of techniques for coordinating agents efficiently and effectively based on multilevel plan models and hierarchical protocols. Some of these main thrusts are very briefly summarized below. Overviews of this work have appeared elsewhere (Durfee, 2001), (Durfee, 2002a), (Durfee, 2002b), (Durfee, 2003).

2.1 Multilevel Modeling

Building from our initial conception of exploiting the structure inherent in hierarchical plan representations, as exemplified by HTNs (Hierarchical Task Networks) and exploiting in systems like PRS (the Procedural Reasoning System), a fundamental challenge that we faced was in capturing the relevant information needed for coordination at the abstract plan levels, based on the ways in which the abstract plans could potentially be refined. We developed sound methods for generating *summary information* by propagating the conditions associated with alternative plan refinements upward in the abstraction hierarchy (Clement, 1999b), (Clement and Durfee, 2000a), (Clement and Durfee, 2000b). A side benefit of these advances is that they provide information that proves to be extremely useful in improving tractability for single-agent planning as well (Clement et al, 2001a), (Clement et al, 2001b).

2.2 Conflict Detection and Resolution

Armed with more complete hierarchical models, agents can engage in a dialogue to discover potential conflicts and resolve them. Our research efforts designed this protocol, which works from more abstract levels downward to focus the search for conflicts and resolutions only on the subportions of the agents' plans that could potentially interact (Clement and Durfee, 1999a), (Clement and Durfee, 1999c). This results in potentially significant streamlining of coordination, permitting the scale up to larger numbers of loosely-interacting agents (Clement and Durfee, 2000c), (Clement, 2002).

In addition, an analysis of potential interactions among agents' primitive actions can lead to the development of a constraint network representing constraints on the overlapping and/or sequential pursuit of actions by multiple agents (Pappachan, 2002). By imposing the hierarchical structure on top of this constraint network, agents can incrementally refine their plans and, at runtime, detect and resolve potential conflicts during dynamic operation (Pappachan and Durfee, 2000), (Pappachan and Durfee, 2001).

2.3 Improving Coordination

The concepts developed for detecting and resolving conflicts can also be applied to identifying and exploiting opportunities for cooperation. That is, different agents might be pursuing similar objectives, and could benefit from working together to avoid redundant efforts. Our research as part of this project has made an initial foray into this area, by extending single-agent plan merging techniques to the multiagent case (Cox and Durfee, 2002), (Cox and Durfee, 2003).

We have also introduced several efficiency improvements to deal with greater system dynamics and for repeated agent interactions. For the former, we recognized that, after they have coordinated, agents might need to change their plans because of changes to the environment or the arrival/departure of agents. The trouble is that the effort spent to create the current coordination solution might be completely lost if the methods simply coordinate (from the top-down) as if from scratch. We have defined concepts of "disruption" to prior coordination solutions, and have revised the search methods to try to find less disruptive solutions sooner (Bartold and Durfee, 2003). For situations where agents might engage in repeated encounters, moreover, it would be wasteful to recalculate the same coordination solutions repeatedly. We have developed methods based on case-based reasoning techniques by which agents save previous coordination solutions to reuse under similar future circumstances (Cox et al, 2001).

2.4 Experimental Results

All of the various research results described above have been prototyped and tested in a variety of ways. The hierarchical representations, coupled with the issues of dynamic execution and plan reuse, have been incorporated into the UMPRS (University of Michigan Procedural Reasoning System) architecture (Cox et al, 2001). A more generic instantiation of these results has been the prototype of the Multilevel Coordination Agent (MCA) that was developed and deployed as a service on the CoABS Grid. The MCA was a part of the Coalition Agents Experiment (CoAX) effort, serving to deconflict the plans of coalition partners to prevent catastrophic interactions (such as friendly fire) and exploit cooperative opportunities (transporting various types of cargo) in the simulated coalition operations (Allsopp et al, 2002). Some of these ideas have also been transitioned to space applications, and have been tested in problems involving the coordination of robotic spacecraft (Clement et al, 2001c).

3 Principal Project Accomplishments

3.1 Technology Base

Designed, developed, implemented, and analyzed methods for generating summary information in concurrent hierarchical plans (CHiPs). Proved correctness based on a principled semantics for plan execution. (See, for example, (Clement and Durfee, 1999b).)

Designed, developed, implemented, and evaluated mechanisms for determining plan interrelationships based on summary information. Proved soundness and completeness. Formally analyzed computational complexity of the algorithms, proving that they can be exponentially faster than techniques that do not employ hierarchical plan structures. (See, for example, (Clement and Durfee, 2000b).)

Developed and tested heuristics, including the Expand-Most-Threats-First heuristic, showing that appropriate heuristics can greatly decrease time for coordination between agents and for hierarchical

planning in the single agent case. Implemented these techniques with other planners, while also introducing the capability of modeling metric resource usage. (See, for example, (Clement et al, 2001).)

Devised and prototyped algorithms for preprocessing plan hierarchies to identify temporal dependencies between plans at different levels of abstraction and represent these dependencies in a temporal constraint network. Demonstrated how the temporal constraint network, coupled with efficient constraint propagation methods, can support flexible coordination in dynamic application domains, including the ability to repair plans as needed. The plan repair algorithm was tested against sample plans from the Coalition domain, where online coordination yielded plans that were 112% more reliable, 11% faster, and required 24% less coordination overhead than using offline coordination that cannot adjust to unexpected events. (See, for example (Pappachan and Durfee, 2001).)

Enhanced failure recovery mechanisms to minimize the degree of disruption to existing commitments between agents. This reduces the coordination overhead incurred for instituting new commitments by an average of 39% in some sample test cases. (See, for example, (Bartold and Durfee, 2003).)

Extended the plan deconfliction capabilities to also exploit the hierarchical plan models so as to efficiently discover opportunities for synergistic, cooperative interactions among separately planning agents. (See, for example, (Cox and Durfee, 2003).)

3.2 Software Deliverables

Automated the translation of coordinated plan information between our algorithms and the UM-PRS agent architecture (which was also modified to communicate through the CoABS Grid). The UM-PRS agents can not only automatically have their uncoordinated plans coordinated by the mechanisms, but can also store and reuse the coordinated solutions. (See, for example, (Cox et al, 2001).)

Implemented core coordination technologies into the Multilevel Coordination Agent (MCA), which resides on the CoABS Grid and has been part of the CoABS Grid release. Integrated this component with relevant other components of the Coalition (CoAX) Technology Integration Experiment (TIE) to support deconfliction of, and synergistic interaction between, plans formulated by the British Defence Evaluation and Research Agency (DERA) Master Battle Planner (MBP) and plans of other allied forces. In doing so, augmented MCA to ensure compliance with MBP's firm execution time constraints, developed translator between the plan representations used by MBP and MCA, and worked with CoAX partners to develop realistic, challenging demonstration scenarios for the 2001 and 2002 demonstrations. (See, for example, (Allsopp et al, 2002).)

3.3 Technology Transition

Coalition (CoAX) TIE: As part of the CoAX TIE, this project integrated its results into that TIE, such that the coalition planning was assured to be conflict free and could exploit serendipitous opportunities for cooperation (Allsopp et all, 2002). There is speculation among colleagues who have participated in real coalition operations that this kind of technology can lead to improvements in coalition planning processes as well as outcomes. As CoAX transitions, as planned, into military applications across multiple branches of the forces (and internationally), this project's coordination technology can transition with it.

NASA-JPL: Members of this project have been engaged in transitioning a number of these ideas into NASA applications, especially in planetary rover technology, in which prototype implementations and evaluations have been conducted (Clement et al, 2001c).

3.4 Student Training

This project supported, in part, the following graduate students:

- Bradley J. Clement (PhD 2002): Abstract Reasoning for MultiAgent Coordination and Planning.
- Pradeep M. Pappachan (PhD 2002): Coordinating Plan Execution in MultiAgent Environments.
- Jeffrey S. Cox (MS 2002, PhD 2005 (expected)): Discovering Synergy Between Hierarchical Planning Agents.
- Thomas Bartold (MS 2002, PhD 2006 (expected)): Limiting Disruption in MultAgent Replanning.
- Haksun Li (PhD 2004 (expected)): Coordination Protocols for Real-Time Agents.

4 Bibliography

Below are papers whose results were, at least in part, supported by this project.

(Allsopp et al, 2002) David N. Allsopp, Patrick Beautement, Jeffrey M. Bradshaw, Edmund H. Durfee, Michael Kirton, Craig A. Knoblock, Niranjan Suri, Austin Tate, and Craig W. Thompson. "Coalition Agents Experiment: Multiagent Cooperation in International Coalitions." *IEEE Intelligent Systems*, 17(3):26-35, May/June 2002.

(Bartold and Durfee, 2003) Thomas Bartold and Edmund H. Durfee. "Limiting Disruption in Multiagent Replanning." To appear in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems,* July 2003.

(Clement and Durfee, 1999a) Bradley J. Clement and Edmund H. Durfee. "Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents." In *Proceedings of the Third International Conference on Autonomous Agents*, pages 252-259, May 1999.

(Clement and Durfee, 1999b) Bradley J. Clement and Edmund H. Durfee. "Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information." In *Proceedings of the National Conference on Artificial Intelligence* (AAAI-99), pages 495-502, July 1999.

(Clement and Durfee, 1999c) Bradley J. Clement and Edmund H. Durfee. "Identifying and Resolving Conflicts among Agents with Hierarchical Plans." In *AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities,* AAAI Technical Report WS-99-12, 6-11, 1999.

(Clement and Durfee, 2000a) Bradley J. Clement and Edmund H. Durfee. "Exploiting Domain Knowledge with a Concurrent Hierarchical Planner." AI and Planning Systems (AIPS-2000) Workshop on *Analysing and Exploiting Domain Knowledge for Efficient Planning, Working Notes*, 57-62, April 2000.

(Clement and Durfee, 2000b) Bradley J. Clement and Edmund H. Durfee. "Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information." In *Proceedings of the 2000 International Workshop on Agent Theories, Architectures, and Languages* (ATAL-2000), pages 202-216, July 2000.

(Clement and Durfee, 2000c) Bradley J. Clement and Edmund H. Durfee. "Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information." (abstract)

Proceedings of the Fourth International Conference on MultiAgent Systems, pages 373-4, July 2000.

(Clement et al, 2001a) Bradley J. Clement, Anthony C. Barrett, Edmund H. Durfee and Gregg R. Rabideau. "Planning with Resources at Multiple Levels of Abstraction." Proceedings of the IJCAI-01 Workshop on Planning with Resources, August 2001.

(Clement et al, 2001b) Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee. "Using Abstraction in Planning and Scheduling." In *Proceedings of the Sixth European Conference on Planning* (ECP-01), pages 145-156, September 2001.

(Clement et al, 2001c) Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee. "Using Abstraction to Coordinate Multiple Robotic Spacecraft." *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2140-2147, October 2001.

(Clement, 2002) B. J. Clement. Abstract Reasoning for MultiAgent Coordination and Planning. PhD Thesis, May 2002.

(Cox et al, 2001) Jeffrey S. Cox, Bradley C. Clement, Pradeep M. Pappachan, and Edmund H. Durfee. "Integrating Multiagent Coordination with Reactive Plan Execution." (abstract) *Proceedings of the ACM Conference on Autonomous Agents (Agents-01)*, pages 149-150, June 2001.

(Cox and Durfee, 2002) Jeffrey S. Cox and Edmund H. Durfee. "Discovering and Exploiting Synergy Between Hierarchical Planning Agents." In *Proceedings of the AAAI 2002 Workshop on Planning With and For MultiAgent Systems*, July, 2002.

(Cox and Durfee, 2003) Jeffrey S. Cox and Edmund H. Durfee. "Discovering and Exploiting Synergy between Hierarchical Planning Agents." To appear in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems,* July 2003.

(Durfee, 2001) Edmund H. Durfee. "Scaling Up Agent Coordination Strategies." *IEEE Computer* 34(7):39-46, July 2001.

(Durfee, 2002a) Edmund H. Durfee. "Strategies for Discovering Coordination Needs in Multi-Agent Systems." *The DoD Software Tech News*, 5(1):3-8, January 2002.

(Durfee, 2002b) Edmund H. Durfee. "Strategies for Discovering Coordination Needs in Multi-Agent Systems." In M. d'Inverno, M. Luck, M. Fisher, and C. Preist (eds.), *Foundations and Applications of Multi-Agent Systems: UKMAS Workshops 1996-2000 Selected Papers*, pages 19-26, Springer-Verlag Lecture Notes in AI 2403, Berlin 2002. (Originally appeared in *The DoD Software Tech News*, 5(1):3-8, January 2002.)

(Durfee, 2003) Edmund H. Durfee. "Challenges to Scaling Up Agent Coordination Strategies." To appear in T. Wagner (ed.), *Multi-Agent Systems: An Application Science*, Kluwer Academic Publishers, 2003.

(Pappachan and Durfee, 2000) Pradeep M. Pappachan and Edmund H. Durfee. "Interleaved Plan Coordination and Execution in Dynamic Multi-agent Domains." (abstract) *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 425-6, July 2000.

(Pappachan and Durfee, 2001) Pradeep M. Pappachan and Edmund H. Durfee. "A satisficing multiagent plan coordination algorithm for dynamic domains." (abstract) *Proceedings of the ACM Conference on Autonomous Agents (Agents-01)*, pages 151-152, June 2001.

(Pappachan, 2002) P. M. Pappachan. Coordinating Plan Execution in Dynamic MultiAgent Environments. PhD Thesis, May 2002.

5 Appendices

The following papers are attached to this report.

(Durfee, 2001) Edmund H. Durfee. "Scaling Up Agent Coordination Strategies." *IEEE Computer* 34(7):39-46, July 2001.

This paper summarizes many of the challenges to coordination, and coalesces a number of the thoughts that arose from discussions among CoABS group members in the first year or two of the project.

(Durfee, 2002a) Edmund H. Durfee. "Strategies for Discovering Coordination Needs in Multi-Agent Systems." *The DoD Software Tech News*, 5(1):3-8, January 2002.

This paper presents a high-level view of the basic concepts behind this project, intended for the DoD Software community.

(Clement and Durfee, 1999b) Bradley J. Clement and Edmund H. Durfee. "Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information." In *Proceedings of the National Conference on Artificial Intelligence* (AAAI-99), pages 495-502, July 1999.

This paper provides details about the plan representation, semantics, and summarization process.

(Clement and Durfee, 2000b) Bradley J. Clement and Edmund H. Durfee. "Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information." In *Proceedings of the 2000 International Workshop on Agent Theories, Architectures, and Languages* (ATAL-2000), pages 202-216, July 2000.

This paper describes and analyzes the coordination process based on summary information of hierarchical plans.

(Clement et al, 2001b) Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee. "Using Abstraction in Planning and Scheduling." In *Proceedings of the Sixth European Conference on Planning* (ECP-01), pages 145-156, September 2001.

This paper illustrates how a number of the ideas for modeling and deconflicting plans of multiple agents can be used fruitfully in the single agent case as well.

(Pappachan and Durfee, 2001) Pradeep M. Pappachan and Edmund H. Durfee. "A satisficing multiagent plan coordination algorithm for dynamic domains." (abstract) *Proceedings of the ACM Conference on Autonomous Agents (Agents-01)*, pages 151-152, June 2001.

This paper summarizes techniques for using temporal constraint networks to support more dynamic, runtime coordination.

(Cox and Durfee, 2003) Jeffrey S. Cox and Edmund H. Durfee. "Discovering and Exploiting Synergy between Hierarchical Planning Agents." To appear in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems,* July 2003.

This paper extends prior techniques to also identify and exploit unexpected cooperative opportunities (synergies) among independently-planning agents..

(Bartold and Durfee, 2003) Thomas Bartold and Edmund H. Durfee. "Limiting Disruption in Multiagent Replanning." To appear in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems,* July 2003.

This paper describes modifications to the coordination search algorithm to allow it to more quickly zoom in on coordination solutions that limit disruption to agents compared to prior commitments they have made.

(Allsopp et al, 2002) David N. Allsopp, Patrick Beautement, Jeffrey M. Bradshaw, Edmund H. Durfee, Michael Kirton, Craig A. Knoblock, Niranjan Suri, Austin Tate, and Craig W. Thompson. "Coalition Agents Experiment: Multiagent Cooperation in International Coalitions." *IEEE Intelligent Systems*, 17(3):26-35, May/June 2002.

This paper summarizes the activities, including the contributions of this project, with respect to the Coalition Agents Experiment (CoAX) efforts.

Appendix A: Challenges to Scaling-Up Agent Coordination Strategies

Edmund H. Durfee

EECS Department
University of Michigan
Ann Arbor, MI 48109
durfee@umich.edu

Introduction

The notion of deploying "intelligent agents" to do peoples' bidding in environments ranging from marketplaces on the internet to robotic exploration of Mars has recently received much attention and speculation. Meanwhile, exactly what an "agent" is and in what senses a computational agent can behave "intelligently" are still undergoing much debate. Rather than confront such thorny issues head on, this article skirts around most of them to focus more squarely on just one of the central concerns of intelligent agency: coordination.

With few exceptions, if an agent is dispatched to an environment, the odds are that it will share the environment with other agents. Even some proposed strategies for robotic exploration of the planets typically involve sending a team of robots! Thus, a fundamental capability needed by an agent is the ability to decide on its own actions in the context of the activities of other agents around it. This is what we will mean when we refer to coordination. Note that this does not mean that coordination must imply cooperation: an effective competitor will coordinate its decisions to work to its advantage against an opponent, such as a producer of goods timing a product promotion to undercut a competitor. It does not even imply reciprocation: an agent may be coordinating with another who is unaware of it, such as one automobile driver trying to pass a second whose mind is entirely elsewhere.

Without coordination, agents can unintentionally conflict, can waste their efforts and squander resources, and can fail to accomplish objectives that require collective effort. It is therefore no wonder that a variety of strategies for coordination among computational agents have been developed over the years, in an effort to get "intelligent agents" to interact at least somewhat "intelligently."

It does not seem possible to devise a coordination strategy that always works well under all circumstances; if such a strategy existed, our human societies could adopt it and replace the myriad coordination constructs we employ, like corporations, governments, markets, teams, committees, professional societies, mailing groups, etc. It seems like whatever strategy we adopt, we can find situations that stress it to the breaking point. Whenever a coordination strategy is proposed, therefore, a natural question that arises is "How does it scale to more stressful situations?"

In an effort to map the space of coordination strategies, therefore, we need to define at least some of these dimensions in which they might be asked to "scale," and then figure out how well they respond to being stressed along those dimensions. For example, clearly one of the most measurable scaling dimensions is simply the number of agents in the system. Yet, sheer numbers cannot be all there is to it: the coordination strategies employed in insect colonies seem to scale to large numbers of insects, yet they do not seem to satisfy all the needs of large human societies (New York City traffic not withstanding). One of my goals in writing this article, therefore, is to provoke a dialogue about what it means for a coordination strategy to "scale up."

Moreover, as Jennings has suggested, agent-oriented software engineering shows promise for developing complex, distributed systems, but requires the component agents to act and interact flexibly (Jennings 2001). A second goal of this article is therefore to provide some potentially useful starting points for characterizing portions of the space of coordination problems, so as to better understand the capabilities and limitations of strategies developed to support flexible interaction. Toward this end, I'll begin by forming a characterization of the coordination problem space by looking at properties of the agent population, of the task-environment the agents inhabit, and of the expectations about their collective behaviors. I'll then turn to giving a very brief survey of a few (of many) coordination strategies and how they fit into this space. I'll conclude by pointing out gaps in our understanding, and suggest opportunities for progress in the field.

Some Dimensions of Coordination Stress

There are more factors that influence how difficult it is to bring about coordination than can be covered here. Therefore, this article tries to project the richness of this space while also simplifying enough to allow a reader to grasp portions the space. To that end, I'll limit discussion to three dimensions (so as to allow depiction on a 2-dimensional page) along each of the major properties: of the agents, of the task-environment, and of the solution. It should be noted up front that these dimensions are not necessarily orthogonal; in some cases relationships between them are indicated. Nonetheless, treating them as orthogonal can be useful in characterizing the space of coordination challenges.

Agent Population Properties

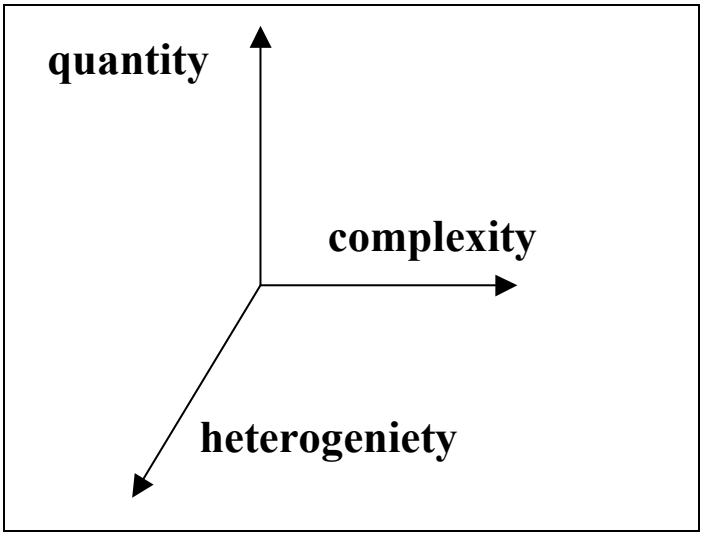
We begin with the most obvious properties that will impact coordination: those of the set of agents that need to coordinate. Certainly, one of the challenges in scaling any coordination strategy, as previously mentioned, is handling larger and larger numbers of agents. Coordination strategies that rely, for example, on a centralized "coordinator" to direct the interactions of the other agents can quickly degrade as the coordinator becomes incapable of processing all of the interactions given increasing numbers of potentially interacting agents. If each agent can potentially interact with every other agent, then the number of pairwise interactions to analyze grows quadratically with the number of agents. More problematically, since interactions often must be viewed in terms of larger groups of agents (not just pairs), the problem can devolve into a problem of exponential size: if each agent could choose among b actions, each potentially having a different impact on other agents, then the space of all possible

action combinations will be b^n, for n agents. Even if each of the n agents participated in the coordination search, rather than depending on a centralized coordinator, an n-fold speedup of a problem that is exponential in n doesn't help much.

A second dimension that often poses challenges in coordination is what is broadly labeled as "heterogeneity." Agents within a population can be different from each other in many possible ways. For example, due to occupying different places in the environment, they might know different things about the current state of the world. If they know different things about the way the world works, then we might say they have heterogeneous expertise. They could have differing abilities to sense the world or change the world. Especially in the case of competitors, they could have different preferences for how the world should be. They could even have different communication languages, ontologies, or internal architectures. Whether a coordination strategy scales to increasingly heterogeneous populations depends on the degree it expects agents to in principle be able to communicate with, share their abilities with, and basically agree with each other.

Finally, the third dimension of agent properties we will consider here is what I term "complexity." While this could mean many things, I'll focus on it as referring to how hard it is to predict what an agent will do because of inherent versatility on the part of an agent. One of the features that arguably makes something an "intelligent" agent is that it is capable of flexibly deciding for itself which goals to pursue at a given time and how to pursue them. Agents that are not complex, under this characterization, are those that can be seen as single-mindedly doing a specialized task. In general, coordinating with such agents is easier (they are much more predictable) than coordinating with agents that could be doing any of a number of things. Couple this with the possibility of overlaps among agents' spheres of interest and ability, and this can put enormous stress on any coordination strategy that that wants to assume unambiguous matches between tasks or roles in the system and the agents to do them.

Obviously, scaling along combinations of these dimensions can pose even greater challenges. Handling complex agents is much harder, for example, if they are complex in different (heterogeneous) ways, but easier if there aren't very many of them. Coordination strategies will tend to therefore make assumptions about which dimensions are likely to be stressed for the application domain of interest.

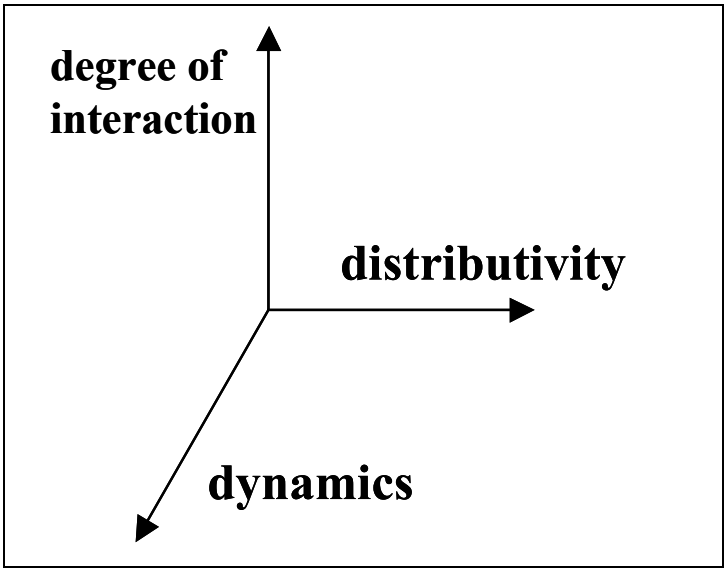


Task-Environment Properties

The environment in which agents operate, and the tasks they are expected to accomplish within the environment, are another major consideration in developing or choosing a coordination strategy. Real task-environments often introduce complications that blur the understanding of a coordination strategy: for example, in task-environments that require substantial domain expertise, it can be difficult to compare alternative coordination strategies because the differences in performance might be due to the quality of the knowledge given the individuals rather than to the efficacy of the coordination strategy. For this reason, researchers often employ abstract, idealized versions of task-environments such as pursuit problems, transport problems, the Prisoners' Dilemma, and distributed sensor networks (e.g., see (Weiss, 1999) and some of the sidebars associated with this article). Even with abstract task-environments, the possible dimensions for scaling the difficulty of coordination are numerous; again, only three are given here of the many possibilities.

The first dimension we will consider is the degree to which the environment, or the task, leads to interactions among agents that materially impact the agents. Since coordination is all about exerting some control over interactions, a greater degree of interaction implies more need to coordinate. Or, viewed the other way, agents that do not interact need not coordinate. Thinking slightly more concretely, suppose that an interaction involves some "issue" that involves more than one agent. The issue could be about who gets to use a resource, or about what the status of some feature of the world is, or about who is supposed to do what task, etc. The degree of agent interaction increases as more agents are concerned with the same issues, and as more issues are of concern to each agent, so that settling some issues commit agents to interactions that in turn impact how they should settle other issues. As the web of dependencies grows, some coordination strategies can have difficulty scaling.

A second dimension that complicates coordination is the dynamics of the task-environment. Coping with changing environments is always difficult; in a multiagent setting, where different agents might be capable of monitoring only portions of the environment, and where each might change its mind about what goals to pursue or what means to use to pursue goals, the difficulties are compounded. In more static task-environments, the agents have some hope of converging on coordinated activities



and then carrying them out. But in more dynamic task-environments, convergence might be impossible: the task-environment might change faster than the coordination strategy can keep up. For this reason, coordination strategies that scale to highly dynamic task-environments are relatively uncommon.

A third dimension, which is related to the first two, as well as to agent heterogeneity, is what here will be called "distributivity." In some task-environments, agents are highly distributed in the (conceptual) environment and tasks are inherently distributed among the agents. In other task-environments, the agents are (conceptually) collected together – such as occupying a common "yellow pages," and tasks originate at one point. Distributivity stresses a coordination strategy because it increases agents' uncertainty about which agents are currently sharing the task-environment and what (if anything) each is, or should be, doing.

Again, scaling along combinations of these dimensions is possible, placing even more substantial demands on a coordination strategy. For example, distributivity compounds the difficulties in a dynamic task-environment, because of the inherent delays in propagating the implications of changes in a highly distributed setting, but lowering the degree of interaction can simplify this by localizing the need to propagate to fewer interested parties.

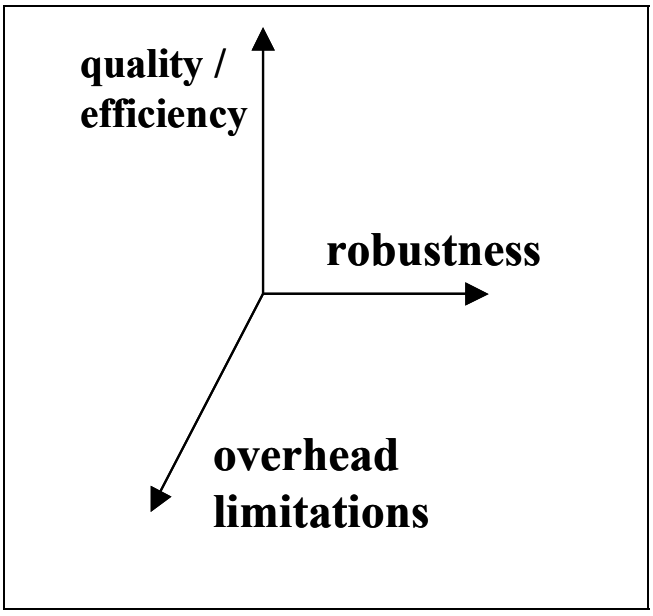
Solution Properties

To evaluate how well a coordination strategy deals with the scaling issues that we throw its way, we need to define criteria that we expect of a solution. One of the dimensions for solution properties, for example, is the "quality" of the solution, in terms of how well the interaction is coordinated. The quality might be measured in terms of efficiency – that is, whether the issues have been settled in a manner that permits the efficient use of agent resources and abilities. Higher quality can correspond to closer to optimal coordination. A less demanding level of quality might correspond to achieving a satisficing level of coordination. In some cases, simply avoiding disagreement (conflict) might be good enough. As illustration, if we were to design a coordination strategy for an automobile intersection, we might be satisfied if it prevents crashes, or we might further require that it achieve some measures such as ensuring no car needs to wait longer than some upper bound time, or we could insist that it minimize the expected wait time for all cars. As we demand more, we put greater stress on the coordination strategy.

A second dimension considers how robust we expect a solution to be in the face of uncertainty or dynamics in the task-environment and the agent population. For example, as was pointed out before, a coordination strategy might have trouble keeping up with a particularly dynamic task-environment. The coordination solution might therefore be somewhat out of date. If we demand that a solution nonetheless be robust, then the coordination strategy should anticipate, either implicitly or explicitly, the range of conditions under which the solution it provides will be followed, and not simply the single expected situation. Given that some task-environments might be such that a minor deviation from expectations can lead to severe consequences, finding assured robust solutions can, in some cases, be imperative.

Finally, a third dimension concentrates on the cost of the coordination strategy. A solution to the problem of how to coordinate should account for the costs of doing the coordination. These costs could include the amount of computation required, communication overhead, time spent, and so on. For example, if communication is costly and time-consuming, a coordination strategy might have to reduce its demands for information exchange among agents; beyond some point, it will have to make high-quality coordination decisions lacking information it would otherwise have expected to have. Therefore, questions can arise about whether a coordination strategy can scale well to environments that impose more stringent limits on costs that the strategy incurs.

As for the previous properties, these three dimensions can combine in various ways. For example, one way of improving robustness of a coordination solution without sacrificing quality is to continually monitor and update the solution in response to changes, but this in turn would require that minimizing costs and delays is not a significant objective.



Characterizing Coordination Strategies

At this point, I've identified three major types of properties (agent population, task-environment, and solution), and for each I've described three (out of many possible) dimensions in which the property could be scaled to make coordination harder. If we were to qualitatively consider "low" and "high" values along each of the dimensions, we'd have eight possible points to consider for each property, leading to 8^3 combinations across the three properties. It would be tempting to now look at each of these 512 combinations in turn, and consider which coordination strategies make sense for each.

The truth is, however, that even if this magazine had enough room, and you the reader had enough patience, there isn't sufficient understanding of the entire space of coordination strategies that have (or could have) computational embodiments to fill all of these in. Instead, what follows summarizes just a handful of coordination strategies, highlighting where they fall within this space and the kinds of scaling for which they are particularly well suited. The selection of these strategies should not be viewed as an endorsement that the strategies given are superior to others not given, but rather is based on giving some representative examples across the space.

More Agents

To some people, "scaling up" is equated to being able to handle more agents, and (almost always) handling more agents is harder than handling fewer. Trying to get a large population of complicated, self-interested, and interacting agents to somehow behave efficiently and robustly in a dynamic environment is a tall

order. In fact, typically something has to give: usually, coordination strategies that scale well to large numbers of agents do not deal with many of these other confounding dimensions.

For example, cellular automata often deal with large numbers of entities that typically use rules to react in simple ways to their very local environments, such as "deactivating" when too few neighbors are active, or "activating" when enough neighbors are active. Patterns of activity can emerge in the population through these very simple local changes. Physics-based models of large computational ecosystems of agents can even lead to designs of metamorphic robots made up many small pieces that shift and flow to adapt to the environment (Bojinov, 2001). Similarly, systems based on insect metaphors assume that each agent is a relatively simple automaton, and that emergent properties of interest arise due to their local interactions (Ferber 1999). These strategies assume little complexity or, often, heterogeneity in the agent population, focus on very limited (local) kinds of interactions, and are satisfied with emergent, statistical system performance, rather than worrying about each agent being efficiently used or making optimal choices.

More generally, successfully scaling up to large numbers of agents generally requires that each agent only needs to interact with a constant (or slowly growing) number of other agents, and that who needs to interact with whom is preordained based on agents' features such as their physical locations or their tasks/roles. Thus, large numbers of mobile agents can be dispersed for information gathering tasks that can be pursued independently, interacting only indirectly due to contention for bandwidth or server cycles (Gray, 2001). Similarly, large-scale coalition/congregation formation can be viewed as an emergent process involving growing groups incrementally as agents (and agent groups) encounter each other and discover advantages of banding together (Lerman, 2000; Brooks, 2000).

More Heterogeneity

In the case of scaling up to large agent populations, agent heterogeneity can sometimes help, if agents that are different from each other need not interact. This serves to once again restrict the number of others about which an agent must be aware. More typically, however, heterogeneity is welcomed into a system because it increases the system-wide capabilities, whereby agents with complementary attributes combine their efforts toward objectives beyond what they can individually achieve. Once the agent population is no longer homogeneous, therefore, it becomes important for agents to be able to understand and often describe what they can do, and to find others with whom to work. Coordination strategies that do not support the ability of agents to describe themselves and to find each other, such as by having implicit acquaintanceships among agents "hardwired," have difficulty scaling along the heterogeneity dimension.

A mainstay coordination strategy for handling heterogeneity has been the Contract Net protocol (Smith 1980) and its descendents, whereby agents dynamically assign tasks to others who are available and capable of doing the tasks. In its simplest form, the protocol allows an agent with a task that it needs done to broadcast an announcement of the task, along with criteria by which each of the other agents can decide whether it is eligible to take on the task and, if so, what information to supply in a bid for the task. The agent with the task can choose from among the responses to make an assignment.

The Contract Net protocol scales well to an open system of heterogeneous agents, but as the number of agents increases, the broadcast communication requirements can be problematic. A response to this is to maintain a more centralized registry of agents and their capabilities, which can be used flexibly to discover promising matches between agents with tasks to do and agents that can do them. Strategies that support agent registration and matchmaking (for example, (Paolucci, 2000) or www.sun.com/jini) can allow agents to find each other by describing the kinds of services that they need or provide. More generally, formalisms for communicative acts, such as FIPA (www.fipa.org), can permit a broad array of conversation policies in support of flexible agent interactions among heterogeneous agents. Many of these concepts are being brought together in more comprehensive frameworks for supporting heterogeneous agent-based systems, such as DARPA's Grid (coabs.globalinfotek.com).

More Complexity

Heterogeneity tends to emphasize the challenges that accrue when "specialist" agents need to identify each other and team to provide broader services. Additional complications arise when agents are individually more complex, typically meaning that they are each more versatile, yet not identically so. Now, each agent must decide which of the possible roles that it could play it should play, and must reason about other agents in terms of the alternative activities they might be engaged in, rather than the specific activity that a "specialist" could be assumed to pursue.

Scaling up to more complex agents means that teaming involves not only finding an available agent with appropriate capabilities, but also selecting from among such agents so as to pick the one whose other talents are least in demand by other teams. Thus, interactions among agents are not localized within smaller teams, but rather the "partial substitutability" of agents for each other leads to complex chains of dependencies: how some teams are formed can color which other teams will be desirable. This means that agents must be increasingly aware of the broader needs of the agent network.

Similarly, even when agents do not need to team up, but merely must co-exist and stay out of each others' way, the increased versatility of each agent makes anticipating what others will be doing much more difficult. Being prepared for anything that another could choose to do might be impossible, so strategies for increasing awareness about other agents' planned activities becomes paramount. Strategies can include using statistics of others' previous behaviors, using observations of them to infer their current plans, or using communication to convey information that permits agents to adequately model each others' intentions.

As an example of the latter, the process by which agents that can accomplish their objectives in several different ways can converge on mutually compatible plans can be viewed as a distributed constraint satisfaction process. This process involves propagating tentative plan choices among agents and, when inconsistencies are detected among the choices of some subset of agents, systematic backtracking is performed by some of the agents. Increased efficiency in this process can stem from techniques that allow parallel asynchronous exploration of the space, and that can dynamically decide which agents should be asked to try alternatives based on measures of which constraints are proving most difficult to satisfy (Weiss, 1999, chapter 4).

Higher Degree of Interaction

As was previously stated, the need for coordination arises from agent interactions. As the number and complexity of agent interactions grow, coordination becomes intractable. Therefore, it isn't surprising that on effective means for addressing coordination is to reduce, or if possible eliminate, interactions. As already pointed out, when agents only have to worry about interactions with a small number of local "neighbors," then scaling to large numbers of agents is much easier. So strategies

for localizing interactions can obviate the need for more complicated coordination strategies.

One often-used technique for controlling the degree of interaction is to impose a (relatively static) organizational structure on agents. Each agent is given a role to play in the organization, including its own sphere of control and knowledge of agents playing related roles. Giving each agent the resources it needs to fulfill its role eliminates the need for agents to negotiate over resources, and giving each agent knowledge of the roles of other agents dictates who needs to communicate with whom and about what. An appropriate organizational structure among agents can thus simplify coordination, and permit larger, more complex agent systems to succeed in more challenging task domains. The challenge, of course, is in designing organizations for agents, or having agents design their own organizations, such that the organizations match the agent population and the needs of the task-environment (Prietula, 1998).

Sometimes, however, multiagent tasks cannot be divided into nearly-independent pieces; there are some tasks that absolutely require tight interactions among agents. In the literature, examples of such tasks include the "pursuit" task where predators need to surround a prey (see sidebar), and tasks involving team activities such as combat flight operations (Tambe, 2000). For such applications, interactions are not a side-effect of individuals acting in a shared world, but rather are the purpose of the individuals' actions in the first place. Therefore, an emphasis on agent *teams* is appropriate, leading to frameworks where a system designer explicitly describes recipes for team behavior, with particular attention to which team members should interact, when, and how (Grosz, 1996; Tambe, 2000).

When agents must formulate plans that fit together, but for which no existing recipes are available, techniques for reasoning about how actions of agents can enable or facilitate, or can hinder or even disable, actions of others, are needed (Decker, 1995). Merging the plans of agents, formulated individually, so as to permit the agents to successfully accomplish their activities without interfering with each other is also a useful technique (Ephrati, 1995; Clement, 1999).

More Dynamic

Whether viewed as a population of individuals or as a team, a multiagent system that operates in a dynamic task-environment must contend with changes in plans, goals, and conditions in the midst of execution. Tasks that previously could be carried out independently might now interact, such as when a resource becomes unusable forcing contention for other remaining resources. Agreements that have been forged between team members might have to be revisited as some team members change their priorities or recognize that their individual intentions, or those of the team as a whole, are no longer relevant in the new context they find themselves in.

Jennings (Jennings, 1992) has characterized these issues as the challenge in having conventions about what agents should do when they begin to question their commitments due to task-environmental dynamics. A variety of conventions can be specified, including the convention that seeks to ignore dynamics entirely by insisting that agents fulfill their commitments regardless. Alternatives include allowing agents to reneg on commitments if they pay some penalty, or permitting agents to abandon obsolete commitments provided that they notify team members (and thus potentially stimulate to formation of different commitments).

In fact, dynamic task-environments can suggest that agents should never view their (or others') plans as being anything more than tentative. Agents could unilaterally change their minds about their plans and begin acting on new plans before reaching agreement across the team. This has the potential of leading to inefficient collective activities due to information delays and to chain reactions (even race conditions) among changes. However, under some limiting assumptions about how and when agents can make unilateral changes, iterative coordination and execution techniques (Weiss, 1999, chapter 3) can lead to flexible coordinated behavior in dynamic task-environments.

More Distributed

Even when the interactions between agents requiring coordination are few and not undergoing dynamic changes, a task-environment can stress agents if the interactions requiring coordination are hard to anticipate. In particular, if agents are acting based on privately-held information about goals and methods, then it might take substantial effort to discover who is going to be interacting with whom.

One response to this is to anticipate all of the possible actions that agents might take, across all of the goals and plans that they might adopt, and to impose restrictions on what actions they can take under what conditions so as to prohibit undesirable interactions. Such "social laws" ensure that a law-abiding agent, acting in a population of other law-abiding agents, need never worry about undesirable interactions, no matter what goals and plans are being adopted (Shoham, 1994). In human terms, this is like saying that as long as all drivers obey traffic laws, then they can each eventually get to their desired destinations, wherever those are, without collision.

A second response is to support the process by which agents whose individual actions might interact can efficiently find each other. When interactions are over the exchange of goods, for example, providing agents with loci (auctions) for finding each other helps. Creating agents to represent resources over which agents might contend similarly allows interacting resource demands to be identified. Or agents might discover through experience others with whom they tend to interact, and form persistent aggregations (Brooks, 2000; Lerman, 2000).

Without identifiable contexts for aggregating, however, it could be that agents must somehow test for possible interactions against all other agents. This could be done through a centralized "coordinator" who collects together information on all agents, and using its global awareness can inform agents of the potential interactions to watch out for. In such a case, the coordinator should accept only as much information as is absolutely necessary to recognize interactions (Clement, 1999). Alternatively, agents could broadcast information to all others, so that each has sufficient awareness of the global picture. Through iterative exchanges, the overall system can cooperatively achieve its objectives.

Greater Optimality/Efficiency

Coordination that is optimal is generally desirable, though less often feasible. As was mentioned earlier, coordination can sometimes be viewed as a search through the exponential number of combinations of agents' alternative actions to find a "good enough" combination. Whereas sometimes it is enough to find a combination that does well enough (avoids conflicts among agents, or ensures eventually achieving goals), for some applications the optimal solution is sought. Optimality generally requires substantial computation (and sometimes communication) overhead; especially in dynamic task-environments (where optimal can become obsolete before it is carried out) or those with many agents and/or complex interactions, a satisficing or locally-optimal solution is often acceptable.

Nonetheless, for some restricted types of coordinated decisions, optimal might be within reach. An example commanding much

attention in recent years has been in coordinating resource allocation decisions based on market-oriented approaches (Weiss, 1999, chapter 5). Through iterated rounds of bidding in an auction, agents can balance supply and demand to allocate resources to maximize their efficient use, under some assumptions. Active research is ongoing to extend these coordination strategies to "scale" them along other dimensions: not only to handle larger numbers of agents, but to handle higher degrees of interaction (using combinatorial auctions to allocate resources whose values are dependent on how they are acquired in combinations) and greater dynamics (including strategies for clearing auctions without waiting for all prices to settle) (Fujishima, 1999).

Other methods for distributed rational decision making (Weiss, 1999, chapter 5) include decision theoretic methods based on multiagent extensions of Markov Decision Processes (Boutilier, 1999). This type of method can find an optimal policy for a multiagent system, based on a particular coordination protocol that can be employed at runtime (for example, to increase agents' awareness of the global situation). When each agent follows its portion of the optimal policy, the expected utility of the multiagent system is maximized.

More Robustness

An optimal coordination solution might break when the world deviates from the coordination strategy's assumptions. Whether a coordination strategy can scale to domains where robust performance is difficult but necessary can thus become important.

One means of increasing the robustness of a coordination solution is to build a solution that contains sufficient flexibility that agents can work around new circumstances within their original coordination agreement. For example, building slack time into scheduled activities, or avoiding committing to details of exactly what will be done and when, can leave each agent with more room to maneuver when the world doesn't proceed according to plan. Typically, more robust coordination decisions are less efficient because they reserve resources for "fall-back" contingencies and therefore might suboptimally divide up tasks among agents for a particular situation. Coordination through organizational structures typically has this feature (Weiss, 1999, chapter 7; Prietula, 1998, chapter 3).

Alternatively, a coordination strategy might expect to monitor the execution of its solution, and repair that solution as needed. These ideas are extensions of single-agent plan monitoring and repair/replan techniques. Teamwork models, with conventions as to how to respond when continued pursuit of joint commitments is senseless, are examples of this (Kumar, 2000). Moreover, in some cases it might be possible to develop generic monitoring and recovery methods for the coordination processes themselves (Dellarocas, 2000).

Lower Overheads

In application domains where communication channels are limited and where the computational resources available for coordination are minimal demand that attention be paid to reducing the overhead of coordination strategies. As communication bandwidth becomes more limited, for example, coordination decisions must be made without exchanging enough information to maintain a level of global awareness that many strategies might expect.

Techniques that involve the iterative exchange of increasingly detailed information about agents' plans and intentions provide one means of permitting time-constrained coordination, where the communication and computation overheads can be limited at the expense of the quality of the coordination solution (Clement, 1999). Alternatively, agents can choose to continue with outdated but still sufficient coordination decisions to avoid a chain reaction of coordination activities. When communication is at a premium, or might even be impossible, techniques such as using observations to model others, or using reasoning to converge on coordinated decisions (e.g., focal points) can pay dividends (Fenster, 1995).

Sometimes, the availability of coordination resources can be sporadic. Under some coordination regimes, agents can take advantage of opportunities where such resources are plentiful to build more complete models of the roles and contingent plans of each other, that can then be exploited when the agents have moved into situations where further communication and computation to coordinate is unsafe or infeasible (Durfee, 1999; Stone 1999).

Open Challenges

I was initially inspired to write this piece because of what I saw as a trend toward identifying scaling to large numbers of agents as the most important challenge that can be posed to a multi-agent system. My own experience was that it was easy to develop multi-agent systems consisting of hundreds or thousands of agents, so long as those agents could merrily go about their business with no concern about the activities of others. On the other hand, it could be a tremendous challenge to develop a working system made up of only a handful of agents if the degree to which their activities needed to be dovetailed – and the penalty for failing to get the dovetailing exactly right – were both very high. The takehome messages of this article could thus be viewed as: (1) there are many ways to stress a coordination strategy, each of which pose research challenges and opportunities, and (2) there are already a variety of promising ideas out there for designing coordination strategies, that can be computationally realized, for getting agents to work well together under a broad range of circumstances.

The preceding whirlwind tour of some of the coordination strategies, and the kinds of stresses in agent population, task-environment, and solution criteria for which they are suited, should be viewed only as an introduction to the rich body of work that has gone into addressing the challenges of coordination in the many domains where it is needed. Many coordination strategies, and variations of coordination strategies, have been left out of the preceding. Interested readers should refer to recent books on the subject of multiagent systems (for example, (Weiss, 1999; Ferber, 1999)) and to journals such as Autonomous Agents and Multi-Agent Systems (published by Kluwer) and proceedings of conferences such as the *Autonomous Agents* conference and the *International Conference on Multi-Agent Systems*.

I should also emphasize that, in the preceding survey, I was not intending that each coordination strategy be pigeonholed as only addressing issues along one of the dimensions. In fact, most can be scaled along multiple dimensions, but each has its limits. The challenge facing researchers in the field is to develop a better (preferably quantifiable) understanding of exactly how far different coordination strategies can scale along the dimensions laid out, as well as along dimensions that are still being identified as being germane to the application of intelligent agent systems to increasingly challenging problems.

Acknowledgments

As part of DARPA's Control of Agent-Based Systems (CoABS), I have worked with several other researchers to understand the relative strengths and weaknesses of various coordination approaches. This article consolidates my take on

many of those thoughts. While these colleagues should be held blameless for any oversimplifications and misrepresentations in this article, I'd like to acknowledge their contributions to my thinking along these lines. They include Craig Boutilier, Jim Hendler, Mike Huhns, David Kotz, James Lawton, Onn Shehory, Katia Sycara, Milind Tambe, and Sankar Virdhagriswaran. Milind Tambe also provided valuable feedback on an earlier version of this article. This work was supported, in part, by DARPA through Rome Labs (F30602-98-2-0142).

References

Bojinov, H., A. Casal, and T. Hogg. "Multiagent Control of Self-reconfigurable Robots." *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 143-150, IEEE Computer Society Press, July 2000.

Boutilier, C. "Multiagent Systems: Challenges and opportunities for decision-theoretic planning." AI Magazine 20(4):35-43, Winter 1999.

Brooks, C. H., E. H. Durfee, and A. Armstrong. "An Introduction to Congregating in Multiagent Systems." *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 79-86, IEEE Computer Society Press, July 2000.

Clement, B. J. and E. H. Durfee, 1999. "Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents." In *Proceedings of the Third Conference on Autonomous Agents*, pages 252-259, May.

Decker, K. S. "TÆMS: A framework for analysis and design of coordination mechanisms." In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, Chapter 16. Wiley Inter-Science, 1995.

Dellarocas, C. and M. Klein. "An experimental evaluation of domain-independent fault handling services in open multi-agent systems." *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 39-46, IEEE Computer Society Press, July 2000.

Durfee, E. H. "Distributed continual planning for unmanned ground vehicle teams." *AI Magazine* 20(4):55-61, Winter 1999.

Ephrati, E., M. E. Pollack, and J. S. Rosenschein. "A tractable heuristic that maximizes global utility through local plan combination." In *Proceedings of the First International Conf. on Multi-Agent Systems* (ICMAS-95), pages 94-101, June 1995.

Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Harlow England, 1999.

Fujishima, Y., Leyton-Brown, K., and Shoham, Y. "Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches." In *Sixteenth International Joing Conference on Artificial Intelligence (IJCAI-99)*. 1999.

Gray, R. S., D. Kotz, R. A. Peterson, Jr., P. Gerken, M. Hofmann, D. Chacon, G. Hill, and N. Suri. "Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task." Technical Report TR2001-386, Dept. of Computer Science, Dartmouth College, January 2001.

Grosz, B. J. and S. Kraus. "Collaborative Plans for Complex Group Action." *Artificial Intelligence*. 86(2), pp. 269-357, 1996.

Jennings, N. R. "Commitments and Conventions: The foundation of coordination in multi-agent systems." *The Knowledge Engineering Review*, 2(3):223-250, 1993.

Jennings, N. R. "An Agent-based Approach for Building Complex Software Systems." Communications of the ACM 44(4):35-41, April 2001.

Kumar, S., P. R. Cohen, and H. J. Levesque. "The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams." *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 159-166, IEEE Computer Society Press, July 2000.

Lerman, K. and O. Shehory. "Coalition Formation for Large-Scale Electronic Markets." *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 167-174, IEEE Computer Society Press, July 2000.

Paolucci, M., Z. Niu, K. Sycara, C. Domashnev, S. Owens and M. van Velsen "Matchmaking to Support Intelligent Agents for Portfolio Management." In *Proceedings of AAAI2000*.

Prietula, M. J., K. M. Carley, and L. Gasser, editors. *Simulating Organizations: Computational Models of Institutions and Groups.* AAAI Press/MIT Press, Menlo Park, CA, 1998.

Shoham, Y. and M. Tennenholtz. "On Social Laws for Artificial Agent Societies: Off-line design." *Artificial Intelligence* 72(1-2):231-252, 1994.

Stone, P. and M. Veloso. "Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork." *Artificial Intelligence* 100(2), June 1999.

Tambe, M., and W. Zhang. "Towards flexible teamwork in persistent teams: extended report." *Journal of Autonomous Agents and Multi-agent Systems* 3(2): 159-183, June 2000.

Weiss, G., editor. *Multiagent Systems: A modern approach to distributed artificial intelligence.* The MIT Press, Cambridge MA, 1999.

Appendix B: Strategies for Discovering Coordination Needs in MultiAgent Systems

Edmund H. Durfee
Computer Science and Engineering Division
EECS Department
University of Michigan
Ann Arbor, MI 48109
durfee@umich.edu

When multiple computational agents share a task environment, interactions between the agents generally arise. An agent might make a change to some feature of the environment that in turn impacts other agents, for example, or might commandeer a non-sharable resource that another agent desires. When the decisions that an agent makes might affect what other agents can or should decide to do, agents will typically be better off if they coordinate their decisions.

Numerous techniques exist for coordinating decisions about potential interactions. These include appealing to a higher authority agent in an organizational structure, instituting social laws that avoid dangerous interactions, using computational markets to converge on allocations, explicitly modeling teamwork concepts, using contracting protocols to strike bargains, and iteratively exchanging tentative plans until all constraints are satisfied. There is a rich literature on these and other mechanisms for coordinating agents; the interested reader can see (Weiss, 1999).

However, each of these mechanisms takes as its starting point that the agents requiring coordination know, at the outset, either with whom they should coordinate, or what issues they should coordinate about. As examples, an organizational structure inherently defines how agents are related to each other, and a computational market corresponds to some resource or "good" that was somehow known to be contentious.

A central thrust of our research is in pushing back the boundaries of what is assumed known in a multiagent setting in order to bootstrap the coordination process. That is, we want to develop techniques by which agents can discover whom they should coordinate with, or what they should coordinate about, so that the rich variety of coordination techniques can then be employed. This paper briefly summarizes some of our progress, results, and plans on this front.

Unintended Conflicts

An important case in which agents need to discover coordination needs is the following. Agents occupy an open, dynamic environment, and each agent has its own independent objectives. Yet, in pursuing its objective, an agent can unintentionally interfere with others, sometimes catastrophically. Therefore, it is important for each agent to discover whether something it is doing needs to be coordinated with others.

We have been studying coalition operations as an example application domain where this kind of problem arises. In a coalition, objectives and responsibilities are distributed among multiple functional teams, where operational choices by one team can infrequently and unintentionally affect another team. The repercussions of unintended interactions can range from merely delaying the accomplishment of objectives (such as waiting for assets that were unexpectedly borrowed by someone else) to more catastrophic outcomes (such as so-called friendly fire). We have been developing computational techniques in which each team is represented by a computational agent, and these agents predict the unintended interactions and resolve them before they occur. The resulting coordinated plans of the agents should be efficient (e.g., agents should not have to wait unnecessarily for others), flexible (e.g., agents should retain room in their plans to improvise around changing local circumstances), and realizable (e.g., agents should not have to message each other at runtime in a manner that outstrips communication capabilities).

Conceptually, our techniques begin by assuming that each agent can represent its plans in a hierarchical task network (HTN), capturing the possible decompositions of abstract plan steps into more detailed plans. As a simple example, consider the case of agent A moving through a grid world to reach a destination (Figure 1). The HTN for this agent is in Figure 2. At the most abstract level (blue arrow in Figure 1, blue node in the HTN), the plan is simply to go from the initial location to the destination. This is in turn composed of the three sequential steps of going to the door, through the door, and beyond the door (green, purple, and aqua arrows/nodes respectively). The ordering constraints are captured in the HTN (Figure 2) by the arrows labeled "B" for "before." For both the first and last step at this level, there are two ways of accomplishing the step. For example, for getting to the door, the red route or the orange route could be chosen. Each of these in turn can be decomposed into a sequence of two movements; red, for example, is to the right and then down).

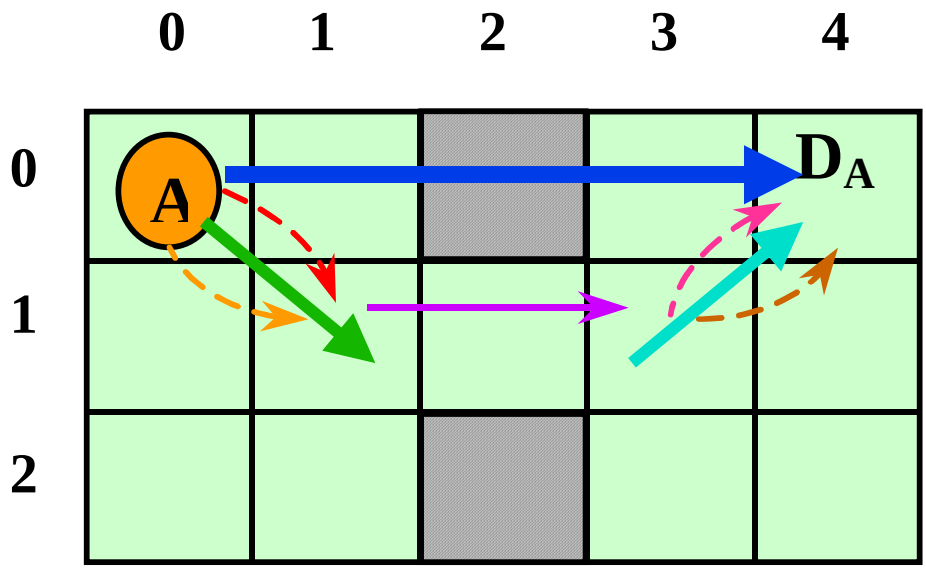


Figure 1: Example Movement Task

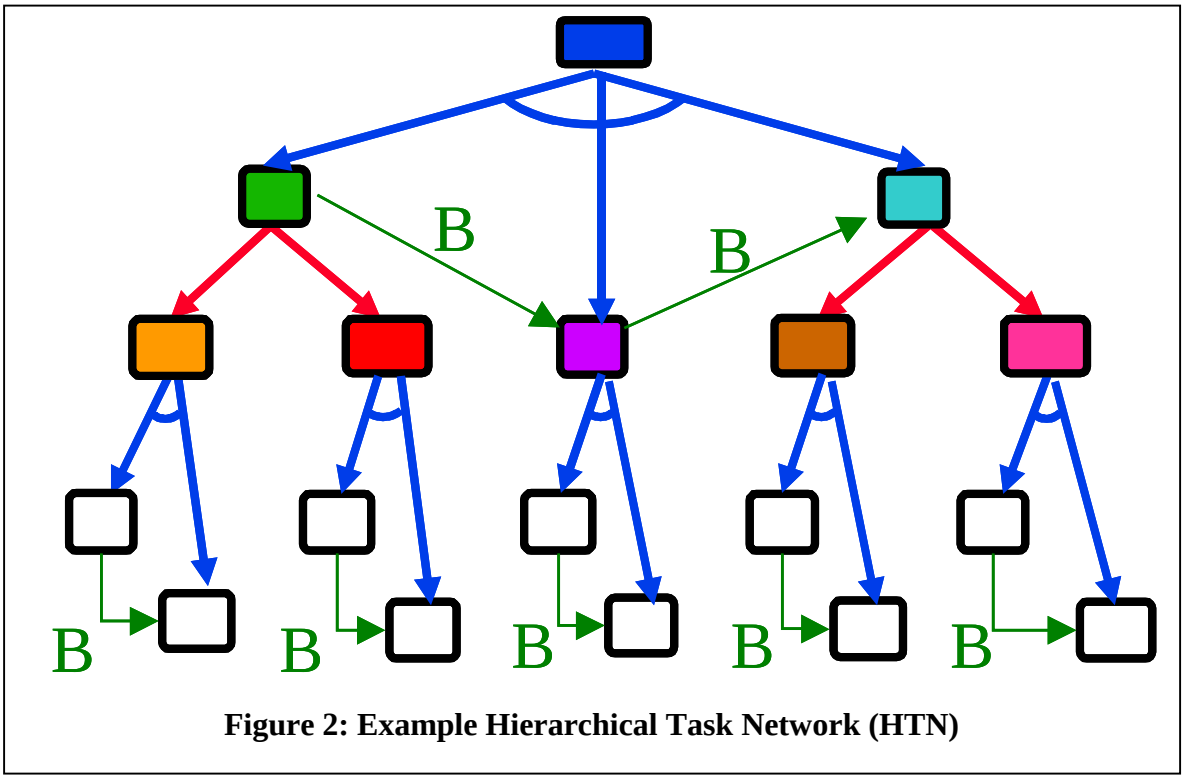


Figure 2: Example Hierarchical Task Network (HTN)

An advantage of using the hierarchical representation is that each agent has, simultaneously, a model of itself at multiple levels of detail. In an open environment populated by numerous agents, being able to communicate about and exchange abstract information can enable agents to quickly determine which (small) subset of agents in the world they actually could potentially interact with (Figure 3a to Figure 3b). In the simple movement task example, for instance, the grid might be much larger, and the subset of agents is small whose planned movements, even abstractly defined, indicate a potential collision with agent A. For those agents, it might even be possible to impose constraints at the abstract level to ensure against unintended collisions, such as sequentializing the overall plans so that only one of the affected agents moves at a time. Or, for the remaining agents, additional details of the HTNs can be exchanged. As a result, agents that were potentially interacting might be determined to not interact at all, reducing the number of agents further and introducing constraints between only substeps of plans leaving agents to do their other substeps as they wish (Figure 3c). Finally, further investigation might indicate that the potential conflict can be isolated to a particular choice that an agent might make; a commitment by the agent to forbid that choice leaves the plans coordinated without imposing any ordering constraints between the agents' plans at all (Figure 3d).

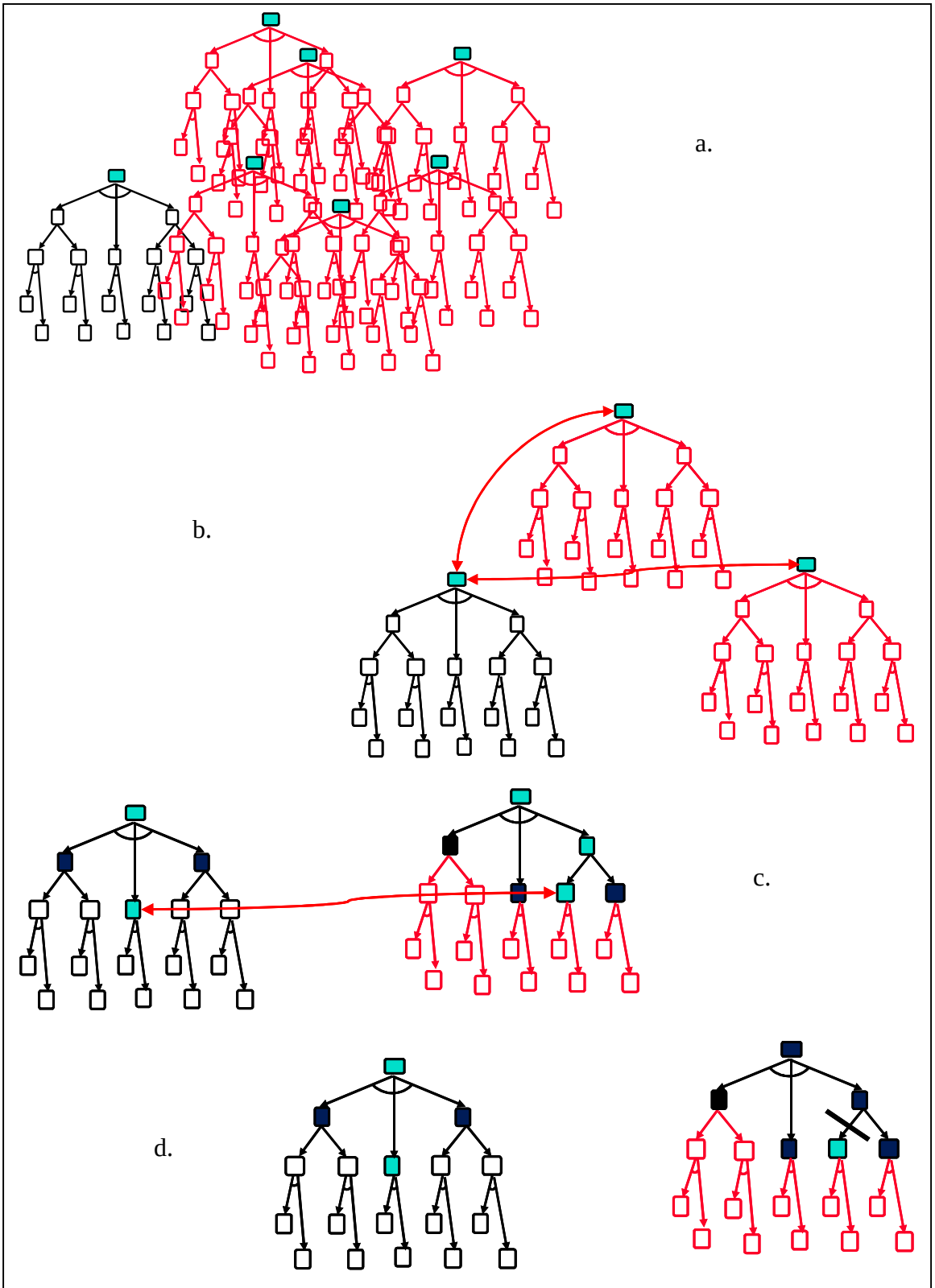


Figure 3: Top-Down Coordination Protocol Example

This example illustrates that, by working from the top down, agents can more efficiently identify and zoom in on the problematic interactions. By digging down deeply, they might be able to impose commitments (on relative timing or on choices of ways in which they will accomplish their tasks) that lead to very crisp coordination. However, in dynamic environments, sometimes it is better to impose constraints at more abstract levels: while this might require more sequential operation than desired, it also allows agents to avoid commitments to details that they might regret. As is intuitive in human coordination, each agent retains more flexibility for improvising when it makes more vague commitments to others. Moreover, digging down deeply requires more rounds of communication and analysis, so coordinating at abstract levels incurs less overhead. Among our ongoing research activities are developing methods for quantitatively evaluating tradeoffs between coordination "crispness", overhead, and flexibility.

We have developed techniques for formulating summaries in HTNs that permit the kind of top-down reasoning that we have just described, and have shown that such techniques can indeed much more efficiently coordinate agents (Clement and Durfee, 1999). These techniques have been shown to be sound and complete. At the cost of completeness, we have also developed a version of these techniques that can be used on-line (Pappachan and Durfee, 2001). The on-line techniques allow agents to postpone decisions about which of the alternative ways they will use to accomplish a task until that task is the next to be done. This in turn provides increased flexibility to the agents, leading to more reliable agent operation in dynamic domains than methods that require agents to make selections before execution begins.

Dealing With Centralization

The techniques just outlined have the feature that, to ensure that all possible interactions are detected and dealt with, some agent or agents need to compare all agents' most abstract plans. This implies that, at some point, information about all agents needs to be known in one place, which is antithetical to decentralized multiagent systems. Certainly, our current implementations rely on a central coordinator to discover potential agent interactions, although in principle once these are discovered the job of working with the agents to resolve the interactions can be delegated to multiple sub-agents, where each handles a different partition of the agent population.

How can we get around the need for centralization? Well, first of all, it should be noted that our use of centralization for detecting interactions does not imply that authority, or even knowledge about agent preferences, is centralized. In our model, the coordination process merely detects potential interactions and finds possible resolutions (more detailed resolutions as time goes on). To agree upon which resolution to use, the affected agents can employ any of the various coordination mechanisms mentioned at the beginning of this paper. That is, these are appropriate once the agents know about the interactions and who is involved.

In turn, this suggests that one way of eliminating the centralization of the detection process requires that agents are initialized with some knowledge. For example, the

organizational structure in which they reside might inherently partition the agents, such that coordination can be carried out in parallel in different partitions. Or agents might be initialized with knowledge of the possible actions of other agents that can be used to anticipate interactions. For example, our research is using these ideas to coordinate resource-limited agents in a multiagent world. In the simplest sense, a resource-limited agent needs to decide how to allocate its limited capabilities in order to meet its performance goals across the scope of worlds that it might encounter. By employing knowledge about what actions other agents might take in particular situations, it can better predict what worlds it might encounter, and can even use its uncertainty to focus communication with those other agents to ask them which of the alternative actions they plan to take for a critical situation. Such communications could also permit agents to avoid taking redundant actions in situations where they would react the same way. In the long run, agents can even engage in negotiation to convince others to favorably change how they react to particular circumstances.

Congregating Over Mutual Concerns

An alternative means of determining coordination needs, instead of centralizing information or inherently distributing key coordination knowledge, is to instead permit coordination needs to be discovered through interactions. While this would be inappropriate in applications where uncoordinated interactions could be catastrophic (such as when friendly fire arises in coalition operations), there are many applications where the consequences of poor coordination are not so dire.

Consider, for example, interactions among groups of people with similar interests, such as in an electronic newsgroup. A well-defined group permits an efficient exchange of relevant information among interested people, with a minimum of tangential communications that waste readers' time. A poorly-defined group, on the other hand, wastes readers time and might lose readership quickly, but with no significant lasting effects on the participants. In this case, then, it is possible that people might congregate around newsgroup topics in an emergent way, through experimentation and exploration in the space, until they converge on relatively stable newsgroups that lead to productive interactions.

We have been conducting research in understanding the dynamic processes of congregating in open environments (Brooks and Durfee, 2000). In our model, agents move among congregations until they find places where they are satisfied, where satisfaction depends on the other members of their congregation. Since these other agents are also moving around to find satisfactory congregations, agents are engaged in non-stationary ("moving target") search. In general, convergence in such systems is slow if it happens at all, and we have been studying mechanisms that enhance convergence such as: varying the movement costs of agents so that some "hold still" while others move; allowing like-minded agents to move as a coalition; giving agents the ability to remember and return to previously-experienced congregations; and allowing agents in a congregation to summarize their common interests and advertise this information to other agents.

As a specific form of congregating, we have been particularly interested in information economies, where competing producers of information goods must bundle and price their goods so as to attract (a subset of) the information consumers. Where a producer ends up in the product-and-price space is influenced not only by the consumer preferences, but also by the positioning decisions of other producers. Among our research results are that we have defined some of the conditions that promote the discovery of niche markets in the information economy, such that producers engage in stable relationships with an interested subset of the consumer population, and avoid mutually-harmful interactions (price wars) with other producers (Brooks, Durfee, and Das, 2000). As suggested above, the price paid for this decentralized technique for discovering which agents should coordinate (interact) with each other is that, on the way to the ultimate mutually-profitable result, producers will sometimes compete with each other and do poorly temporarily as a result.

Summary and Future Directions

In this paper, we have claimed that, while powerful techniques exist for coordinating agents that already know whom or about what to coordinate, there are still many issues that need to be explored in designing efficient mechanisms by which to determine what needs to be coordinated in the first place. We briefly described some mechanisms that we are exploring for this purpose. One of these involves agents iteratively exchanging plan information at increasingly detailed levels to isolate potential interactions and impose effective commitments to resolve conflicts. Another, on the other hand, permits suboptimal interactions to occur, and allows the agent population to self-organize, over time, into congregations that emphasize beneficial interactions.

There are many directions in which we are, or are considering, extending these research activities. We need to develop heuristic means by which agents can decide on the level of detail at which they should coordinate, and metrics for comparing alternative coordination decisions in uncertain environments. We need to extend the soundness and completeness proofs, as well as the complexity analyses, of the techniques as we continue to augment and improve them. Coordination commitments that are derived between agents should be generalized and remembered to form the core of a suite of team plans, and the processes by which coordination needs are discovered should apply not only between agents but also between agent teams. Finally, these techniques need to be implemented and evaluated in the context of challenging applications, such as in the domain of coordinating coalition operations.

Acknowledgements

The ideas and results described in this paper were developed with numerous collaborators. In particular, I'd like to thank my students, including Brad Clement, Pradeep Pappachan, Chris Brooks, Haksun Li, and Jeff Cox. The work was supported, in part, by DARPA under the Control of Agent-Based Systems Initiative (F30602-98-2-0142), by DARPA under the Automated Negotiating Teams Initiative (subcontract to Honeywell on F30602-00-C-0017), and by NSF grant IIS-9872057.

References

Christopher H. Brooks, Edmund H. Durfee and Aaron Armstrong. “An Introduction to Congregating in Multiagent Systems.” In *Proceedings of the Fourth International Conference on MultiAgent Systems* (ICMAS-2000), pages 79-86, July 2000.

Christopher H. Brooks, Edmund H. Durfee and Rajarshi Das. “Price Wars and Niche Discovery in an Information Economy.” In *Proceedings of the ACM Conference on Electronic Commerce 2000 (EC-00),* October 2000.

Bradley J. Clement and Edmund H. Durfee. “Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information.” In *Proceedings of the National Conference on Artificial Intelligence* (AAAI-99), pages 495-502, July 1999.

Pradeep M. Pappachan and Edmund H. Durfee. “A satisficing multiagent plan coordination algorithm for dynamic domains.” (abstract) *Proceedings of the ACM Conference on Autonomous Agents (Agents-01),* June 2001.

Gerhard Weiss (editor). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press: Cambridge Massachusetts, 1999.

Author Information

Edmund H. Durfee is currently a Professor of Electrical Engineering and Computer Science, and of Information, at the University of Michigan. He received a Ph.D. degree in Computer Science from the University of Massachusetts in 1987 and joined the EECS Department at the University of Michigan in 1988. His area of teaching and research is artificial intelligence, multi-agent systems, and real-time intelligent control. To date he has published over 100 journal and conference papers, and has served in various capacities such as the program chair (1998) and conference chair (2000) for the International Conference on MultiAgent Systems. He has received a *Presidential Young Investigator Award* from the NSF (1991), is a senior member of IEEE, and has been elected as a *Fellow of the American Association of Artificial Intelligence (AAAI).*

Appendix C:

Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information

Bradley J. Clement and Edmund H. Durfee
University of Michigan
Ann Arbor, MI 48109
{bradc, durfee}@umich.edu

Abstract

Interacting agents that interleave planning, plan coordination, and plan execution for hierarchical plans (e.g. HTNs or procedures for PRS) should reason about abstract plans and their concurrent execution before they are fully refined. Poor decisions made at abstract levels can lead to costly backtracking or even failure. We claim that better decisions require information at abstract levels that summarizes the preconditions and effects that must or may apply when a plan is refined. Here we formally characterize concurrent hierarchical plans and a method for deriving summary information for them, and we illustrate how summary conditions can be used to coordinate the concurrent interactions of plans at different levels of abstraction. The properties of summary conditions and rules determining what interactions can or might hold among asynchronously executing plans are proven to support the construction of sound and complete coordination mechanisms for concurrent hierarchical planning agents.

Introduction

The study of concurrent action in relation to planning (Georgeff 1984) has improved our understanding of how agents can reason about their interactions in order to avoid conflicts during concurrent plan execution. Conflicts can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents (Lansky 1990), and by merging the individual plans of agents by introducing synchronization actions (Georgeff 1983). In fact, planning and merging can be interleaved, such that agents can propose next-step extensions to their current plans and reconcile conflicts before considering extensions for subsequent steps. By formulating extensions in terms of constraints rather than specific actions, a "least commitment" policy can be retained (Ephrati & Rosenschein 1994).

For many applications, planning efficiency can be enhanced by exploiting the hierarchical structure of planning operations. Rather than building a plan from the beginning forward (or end backward), hierarchical planners identify promising classes of long-term activities (abstract plans), and incrementally refine these to eventually converge on specific actions. Planners such as NOAH (Sacerdoti 1977) and NONLIN (Tate 1977) have this character, and are often considered instances of a class of planners called Hierarchical Task Network (HTN) planners. By exploiting the hierarchical task structure to focus search, HTN planners often converge much more quickly to effective plans. They are also becoming increasingly well understood (Erol, Hendler, & Nau 1994).

Using HTN planning for concurrently-executing agents is less well understood, however. If several HTN planning agents are each generating their own plans, how and when should these be merged? Certainly, merging could wait until the plans were fully refined, and techniques like those of Georgeff (mentioned previously) would work. But interleaving planning and merging holds greater promise for identifying and resolving key conflicts as early in the process as possible to try to avoid backtracking or failure. Such interleaving, however, requires the ability to identify potential conflicts among abstract plans.

Corkill (Corkill 1979) studied interleaved planning and merging in a distributed version of the NOAH planner. He recognized that, while most of the conditions affected by an abstract plan operator might be unknown until further refinement, those that deal with the overall effects and preconditions that hold no matter how the operator is refined can be captured and used to identify and resolve some conflicts. He recognized that further choices of refinement or synchronization choices at more abstract levels could lead to unresolvable conflicts at deeper levels, and backtracking could be necessary. Our work is directed toward avoiding such backtracking by improving how an abstract plan operator represents all of the potential needs and effects of all of its potential refinements.

Our motivation for doing this is not simply to make interleaved planning and merging with HTNs more efficient, but also to support another crucial use of HTN concepts–specifically, flexible plan execution systems such as PRS (Georgeff & Lansky 1986), RAPS

This work was supported in part by NSF (IRI-9158473) and DARPA (F30602-98-2-0142).

(Firby 1989), etc., that similarly exploit hierarchical plan spaces. Rather than refine abstract plan operators into a detailed end-to-end plan, however, these systems interleave refinement with execution. By postponing refinement until absolutely necessary, such systems leave themselves flexibility to choose refinements that best match current circumstances. However, this means that refinement decisions at abstract levels are made and acted upon before all of the detailed refinements need be made. If such refinements at abstract levels introduce unresolvable conflicts at detailed levels, the system ultimately gets stuck part way through a plan that cannot be completed. While backtracking is possible for HTN planning (since no actions are taken until plans are completely formed), it might not be possible when some (irreversible) plan steps have already been taken. It is therefore critical that the specifications of abstract plan operators be rich enough to summarize all of the relevant refinements to anticipate and avoid such conflicts. In this paper, we formally characterize methods for deriving and exploiting such rich summaries to support interleaved local planning, coordination (plan merging), and execution.

Simple Example

This example illustrates the use of summary information, explains some terminology, and further motivates the formalism of a theory for concurrent hierarchical plans (CHiPs) and summary information.

Suppose that two agents wish to go through a doorway to another location, (*row*, *column*), as shown in Figure 1. Agent A has a hierarchical plan, p, to move from (0,0) to (0,4), and B also has a plan, q, to move from (2,0) to (2,4), but they need to coordinate their plans to avoid collision. Agent A could have preprocessed plan p to derive its summary information. The set of *summary preconditions* of p includes all its preconditions and those of its subplans that must be met external to p in order for p to execute successfully: $\{At(A,0,0)$, $\neg At(B,0,1)$, $\neg At(B,1,0)$, ..., $\neg At(B,0,4)\}$. The proposition $At(A,0,0)$ is a *must* condition because no matter how p is executed, the condition must hold. $\neg At(B,1,0)$ is *may* because it may be required depending on the path A takes. Likewise, the *summary postconditions* of p are its effects and those of its subplans that are seen externally: $\{At(A,2,0)$, $\neg At(A,0,0)$, $\neg At(A,1,0)$, ...$\}$. The *summary inconditions* are any conditions that must hold within the interval of time that the plan is executing and can be *must* or *may* and *always* or *sometimes*. An *always* condition is required to hold throughout the duration of any execution of the plan. For example, a *must*, *always* incondition of p could be $PowerOn(A)$– the power must always be on. $At(A,1,0)$ is a *may*, *sometimes* incondition of p because A *may* choose that path and would only be there at *some time*. These conditions and descriptors, such as *must* and *always*, provide the necessary information to reason about what conditions must or may be achieved or clobbered when ordering a set of plan executions.

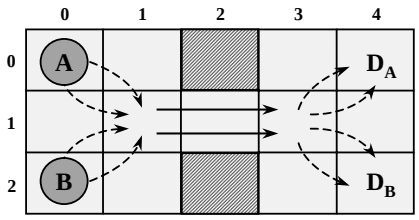


Figure 1: Agents A and B go through a doorway.

Now suppose A sends B p_{sum}, the summary information for p. Agent B can now reason about the interactions of their plans based on their combined summary information. For instance, based only on the summary information, B can determine that if p is restricted to execute before q, then the plans *can* be executed (refined) in *any way*, or $CanAnyWay$(b,p_{sum},q_{sum}).[1] So, B could tell A to go ahead and start execution and to send back a message when p is finished executing. However, B may instead wish to overlap their plan executions for better efficiency. Although $CanAnyWay$(o,p_{sum},q_{sum}) is not true, B could use the summary conditions to determine that there *might* be *some way* to overlap them, or $MightSomeWay$(o,p_{sum},q_{sum}). Then, B could ask A for the summary information of each of p's subplans, reason about the interactions of lower level actions in the same way, and find a way to synchronize the subplans for a more fine-grained solution.

Consider another case where A and B plan to move to the spot directly between them, (1,0), and can choose from different routes. $MightSomeWay$(b, p_{sum}, q_{sum}) would be false since the postconditions of p must always clobber the preconditions of q. If we wanted to describe a rule for determining whether two actions can or might overlap, it is not obvious how this should be done. The difficulty of composing such rules stems from an imprecise specification of concurrent plan execution and the large space of potential plans that have the same summary information. If the $MightSomeWay$(o, p_{sum}, q_{sum}) rule is not specified in a complete way, the agent may not determine that the *overlaps* relation cannot hold until it has exhaustively checked all synchronizations of p and q's primitive subplans. As the number of subplans grows, this becomes an intractable procedure (Vilain & Kautz 1986). Even worse would be if, for the sake of trying to be complete, a rule is specified in an unsound way leading to a synchronization choice that causes failure. We give an example of this in (Clement & Durfee 1999b), where we also implement a hierarchical plan coordi-

[1] We will often abbreviate Allen's thirteen temporal relations (Allen 1983). Here, "b" is for the *before* relation. "o" is for *overlaps*.

nation algorithm that uses summary information in the manner described above. Our evaluations show that coordinating at different levels of abstraction for different cost scenarios results in better performance. Thus, formalizing concurrent hierarchical plans, their execution, and the derivation of summary conditions is necessary to avoid costly, irreversible decisions made during planning, plan execution, and coordination.

Overview

In the next section we describe the semantics of hierarchical plans and their concurrent execution to ground our theory. The simple theory of action consistently describes all temporal interactions among primitive or hierarchical plans. We basically add a set of *inconditions* to popular STRIPS-style plan representations to reason about concurrent plan execution. In addition, we formalize traditional planning concepts, such as *clobbers* and *achieves*, and reintroduce *external conditions* (Tsuneto, Hendler, & Nau 1998) for reasoning about CHiPs. We then describe the semantics of plan summary information and a correct method for deriving it efficiently. This, in turn, is used to describe the construction of sound and complete rules for determining how plans can definitely or might possibly be temporally related. The result is a theory for proving correct coordination and planning mechanisms.

A Model of Hierarchical Plans and their Concurrent Execution

The original purpose of developing the following theory was to provide, as simply as possible, a consistent model of execution to generally reason about the concurrent interaction of hierarchical plans. However, we also wanted the model to share important aspects of plans used by PRSs, HTNs, Allen's temporal plans, and many STRIPS-style plan representations. As such, this theory of action tries to distill appropriate aspects of other theories, including (Allen & Koomen 1983), (Georgeff 1984), and (Fagin *et al.* 1995).

CHiPs

A concurrent hierarchical plan p is a tuple $\langle pre,\ in,\ post,\ type,\ subplans,\ order\rangle$. $pre(p)$, $in(p)$, and $post(p)$ are sets of literals (v or $\neg v$ for some propositional variable v) representing the preconditions, inconditions, and postconditions defined for plan p.[2] The *type* of plan p, $type(p)$, has a value of either *primitive*, *and*, or *or*. An *and* plan is a non-primitive plan that is accomplished by carrying out all of its subplans. An *or* plan is a non-primitive plan that is accomplished by carrying out one of its subplans. So, *subplans* is a set of plans, and a *primitive* plan's *subplans* is the empty set. $order(p)$ is only defined for an *and* plan p and is a set of temporal relations (Allen 1983) over pairs of subplans that together are consistent; for example, $before(p_i, p_j)$ and $before(p_j, p_i)$ could not both be in *order*. Plans left unordered with respect to each other are interpreted to potentially execute in concurrently. For the example in Figure 1, A's highest level plan p is the tuple $\langle\{\},\ \{\},\ \{\},\ and,\ \{m_1,\ m_2,\ m_3\},\ \{before(m_1,\ m_2),\ before(m_2,\ m_3)\}\rangle$. Here, m_1, m_2, and m_3 correspond to p's subplans for moving to (1,1), (1,3), and (0,4) respectively. There are no conditions defined because p can rely on the conditions defined for the primitive plans for moving between grid locations. The primitive plan for moving agent A from (1,3) to (0,3) is the tuple $\langle\{At(A,1,3)\},\ \{At(A,1,3),\ \neg At(B,1,3),\ \neg At(B,0,3)\},\ \{At(A,0,3),\ \neg At(A,1,3),\ \neg At(B,0,3),\ \neg At(B,1,3)\},\ primitive,\ \{\},\ \{\}\rangle$.

We also require postconditions to specify whether the inconditions change or not. This helps simplify the notion of inconditions as conditions that hold only *during* plan execution whether because they are *caused* by the action or because they are *necessary conditions* for successful execution. If a plan's postconditions did not specify the truth values of the inconditions' variables at the end of execution, then it is not intuitive how those values should be determined in the presence of concurrently executing plans. By requiring postconditions to specify such values, we resolve all ambiguity and simplify state transitions (described in the section below on Histories and Runs).

The decomposition of a CHiP is in the same style as that of an HTN as described by Erol *et al.* (Erol, Hendler, & Nau 1994). An *and* plan is a task network, and an *or* plan is an extra construct representing a set of all tasks that accomplish the same goal or compound task. Tasks in a network are subplans of the plan corresponding to the network. High-level effects (Erol, Hendler, & Nau 1994) are simply the postconditions of a non-primitive CHiP. CHiPs can also represent a variety of interesting procedures executable by PRSs.

Executions

We recursively describe an execution of a plan as an instance of a decomposition and ordering of its subplans' executions. This helps us reason about the outcomes of different ways to execute a group of plans, describe state transitions, and formalize other terms.

The possible executions of a plan p is the set $\mathcal{E}(p)$. An *execution* of p, $e \in \mathcal{E}(p)$, is a triple $\langle d, t_s, t_f\rangle$. $t_s(e)$ and $t_f(e)$ are positive, non-zero real numbers representing the start and finish times of execution e, and $t_s < t_f$. $d(e)$ is a set of subplan executions representing the decomposition of plan p under this execution e. Specifically, if p is an *and* plan, then it contains one execution from each of the subplans; if it is an *or* plan, then it contains only one execution of one of the subplans; and it is empty if it is *primitive*. In addition, for all subplan executions, $e' \in d$, $t_s(e')$ and $t_f(e')$ must be consistent with the relations specified

[2] Functions such as $pre(p)$ are used for referential convenience throughout this paper. Here, *pre* and $pre(p)$ are the same, and $pre(p)$ is read as "the preconditions of p."

in $order(p)$. Also, the first subplan(s) to start must start at the same time as p, $t_s(e') = t_s(e)$; and the last subplan(s) to finish must finish at the same time as the p, $t_f(e') = t_f(e)$. An execution for agent A's top-level plan p (described previously in the section on CHiPs) would be some $e \in \mathcal{E}(p)$. e might be $\langle\{e_1, e_2, e_3\}, 4.0, 10.0\rangle$ where $e_1 \in \mathcal{E}(m_1)$, $e_2 \in \mathcal{E}(m_2)$, $e_3 \in \mathcal{E}(m_3)$, and e begins at time 4.0 and ends at time 10.0. e_1 also starts at 4.0, and e_3 ends at 10.0.

The *subexecutions* of an execution e, sometimes referred to as $subex(e)$, is defined recursively as the set of subplan executions in e's decomposition unioned with their subexecutions. For agent A, $subex(e) = \{e_1, e_2, e_3\} \cup subex(e_1) \cup subex(e_2) \cup subex(e_3)$. For convenience, we say that a condition of a plan with an execution in the set containing e and e's subexecutions is a *condition of* e. So, if A executes its top-level plan, since $\neg At(B, 1, 2)$ is an incondition of the primitive for A to move from (1,1) to (1,2), it is also an incondition of the primitive's execution, e_2, and e.

Histories and Runs

We describe hypothetical possible worlds, called histories, so that we can determine what happens in all worlds, some, or none. We then can describe how the state transforms according to a particular history. A state of the world, s, is a truth assignment to a set of propositions, each representing an aspect of the environment. We treat a state as the set of true propositional variables.

A *history*, h, is a tuple $\langle E, s_I\rangle$. E is a set of plan executions including those of all plans and subplans executed by all agents, and s_I is the initial state of the world before any plan is begun. So, a history h is a hypothetical world that begins with s_I as the initial state and where only executions in $E(h)$ occur.

A *run*, r, is a function mapping time to states. It gives a complete description of how the state of the world evolves over time. We take time to range over the positive real numbers. $r(t)$ denotes the state of the world at time t in run r. So, a condition is *met* at time t if the condition is a non-negated propositional variable v, and $v \in r(t)$ or if the condition is a negated propositional variable $\neg v$, and $v \notin r(t)$.

For each history h there is exactly one run, $r(h)$[3], that specifies the state transitions *caused* by the plan executions in $E(h)$. The interpretation of a history by its run is defined as follows. The world is in the initial state at time zero: $r(h)(0) = s_I(h)$. In the smallest interval after any point where one or more executions start and before any other start or end of an execution, the state is updated by adding all non-negated inconditions of the plans and then removing all negated inconditions. Similarly, at the point where one or more executions finish, the state is updated by adding all non-negated postconditions of the plans and then removing all negated postconditions. Lastly, if no execution of a plan begins or ends between two points in time, then the state must be the same at those points. First order logic sentences for these axioms are specified in a larger report (Clement & Durfee 1999a).

Now we can define what it means for a plan to execute successfully. An execution $e = \langle d, t_s, t_f\rangle$ *succeeds in* h if and only if the plan's preconditions are met at t_s; the inconditions are met throughout the interval (t_s, t_f); the postconditions are met at t_f; and all executions in e's decomposition are in $E(h)$ and succeed. Otherwise, e *fails*. So, in a history h where agent A successfully executes a plan (as described previously in the section on Executions) to traverse the room, $E(h) = \{e\} \cup subex(e)$, and all conditions of all plans with executions in $E(h)$ are met at the appropriate times. Given the example primitive conditions in the section on CHiPs and the axioms just described for state transitions, if agent B happened to start moving into A's target location, (0,4), at the same time as A, then either A's primitive plan execution e_A finishes before B's and the $\neg At(A, 0, 4)$ incondition of B's primitive execution e_B is not met (clobbered) at $t_f(e_A)$; e_B finishes before e_A and similarly clobbers e_A's incondition; or they both finish simultaneously clobbering each others' $At(A/B, 0, 4)$ postconditions. If e_A fails, then e_3 and the top-level execution e must also fail.

Asserting, Clobbering, and Achieving

In conventional planning, we often speak of *clobbering* and *achieving* preconditions of plans (Weld 1994). In CHiPs, these notions are slightly different since inconditions can clobber and be clobbered, as seen in the previous section. Formalizing these concepts helps prove properties of summary conditions. However, it will be convenient to define first what it means to *assert* a condition.

An execution e of plan p is said to *assert* a condition ℓ at time t in a history h if and only if ℓ is an incondition of p, t is in the smallest interval beginning after $t_s(e)$ and ending before a following start or finish time of any execution in $E(h)$, and ℓ is satisfied by $r(h)(t)$; or ℓ is a postcondition of p, $t = t_f(e)$, and ℓ is satisfied by $r(t)$. So, asserting a condition only *causes* it to hold if the condition was not previously met. Otherwise, the condition was already satisfied and the action requiring it did not really *cause* it.

A *precondition* ℓ of plan p_1 is [*clobbered*, *achieved*][4] in e_1 (an execution of p_1) by e_2 (an execution of plan p_2) at time t if and only if e_2 asserts $[\ell', \ell]$ at t; $\ell \Leftrightarrow \neg\ell'$; and e_2 is the last execution to assert ℓ or ℓ' before or at $t_s(e_1)$. An [*incondition*, *postcondition*] ℓ of plan p_1 is *clobbered* in e_1 by e_2 at time t if and only if e_2 asserts ℓ' at t; $\ell \Leftrightarrow \neg\ell'$; and $[t_s(e_1) < t < t_f(e_1),$

[3] For convenience, we now treat r as a function mapping histories to runs, so $r(h)(t)$ is a mapping of a history and a time to a state.

[4] We use braces [ ] as a shorthand when defining similar terms and procedures. For example, saying "[a, b] implies [c, d]" means a implies c, and b implies d.

$t = t_f(e_1)$]. Achieving inconditions and postconditions does not make sense for this formalism, so it is not defined. In the previous section when e_A finished first and asserted $At(A, 0, 4)$, it clobbered the incondition $\neg At(A, 0, 4)$ of B's primitive plan in e_B at $t_f(e_A)$.

External Conditions

As recognized in (Tsuneto, Hendler, & Nau 1998), external conditions are important for reasoning about potential refinements of abstract plans. Although the basic idea is the same, we define them a little differently and call them *external preconditions* to differentiate them from other conditions we call *external postconditions*. Intuitively, an external precondition of a group of partially ordered plans is a precondition of one of the plans that is not achieved by another in the group and must be met external to the group. External postconditions, similarly, are those that are not undone by plans in the group and are net effects of the group.

Formally, an *external precondition* ℓ of an *interval* (t_1, t_2) in history h is a precondition of a plan p with some execution $e \in E(h)$ for which $t_1 \leq t_s(e) < t_2$, and ℓ is neither achieved nor clobbered by an execution at a time t where $t_1 \leq t \leq t_s(e)$. An *external precondition* of an *execution* $e = \langle d, t_s, t_f \rangle$ is an external precondition of an interval (t_1, t_2) in some history where $t_1 \leq t_s$; $t_f \leq t_2$; and there are no other plan executions other than the subexecutions of e. An *external precondition* of a *plan* p is an external precondition of any of p's executions. It is called a *must* precondition if it is an external precondition of all executions; otherwise it is called a *may* precondition. $At(A, 0, 0)$ is an external precondition of agent A's top-level plan p (Figure 1) since no subplan in p's hierarchy achieves $At(A, 0, 0)$. $At(A, 1, 1)$ is not an external precondition of p because it is achieved internally by the execution of subplan m_1 (described in the section on CHiPs).

Similarly, an *external postcondition* ℓ of an *interval* (t_1, t_2) in h is a postcondition of a plan p with some execution $e \in E(h)$ for which $t_1 \leq t_f(e) \leq t_2$; ℓ is asserted by e; and ℓ is not clobbered by any execution at a time t where $t_f(e) < t \leq t_2$. External postconditions of executions and plans can be defined in the same way as external preconditions. $At(A, 0, 4)$ is an external postcondition of agent A's top-level plan p since no subplan in p's hierarchy cancels the effect. $At(A, 1, 3)$, an external postcondition of m_2 is not an external postcondition of p because it is cancelled internally by the execution of subplan m_3 when it later asserts $\neg At(A, 1, 3)$.

Plan Summary Information

With the previous formalisms, we can now define summary information and describe a method for computing it for non-primitive plans. The *summary information* for a plan p is p_{sum}. Its syntax is given as a tuple $\langle pre_{sum}, in_{sum}, post_{sum} \rangle$, whose members are sets of *summary conditions*. The *summary* [*pre*, *post*] *conditions* of p, [$pre_{sum}(p)$, $post_{sum}(p)$], contain the external [pre, post] conditions of p. The *summary inconditions* of p, $in_{sum}(p)$, contain all conditions that must hold within some execution of p for it to be successful. A condition c in one of these sets is a tuple $\langle \ell, existence, timing \rangle$. $\ell(c)$ is a literal. The *existence* of c can be *must* or *may*. If $existence(c) = must$, then c is called a *must* condition because ℓ holds for every successful plan execution (ℓ *must* hold). For convenience we usually write $must(c)$. c is a *may* condition ($may(c)$ is *true*) if there is at least one plan execution where $\ell(c)$ must hold. The *timing* of c can take the values *always*, *sometimes*, *first*, *last*. $timing(c)$ is *always* for $c \in in_{sum}$ if $\ell(c)$ is an in-condition that must hold throughout the execution of p (ℓ holds *always*); otherwise, $timing(c) = sometimes$ meaning $\ell(c)$ holds at one point, at least, within an execution of p. The *timing* is *first* for $c \in pre_{sum}$ if $\ell(c)$ holds at the beginning of an execution of p; otherwise, $timing = sometimes$. Similarly, *timing* is *last* for $c \in post_{sum}$ if $\ell(c)$ holds at the end of an execution of p; otherwise, it is *sometimes*. Although *existence* and *timing* syntactically only take one value, semantically $must(c) \Rightarrow may(c)$, and $always(c) \Rightarrow sometimes(c)$. See the Introduction for an example of summary conditions derived for an abstract plan.

Deriving Summary Conditions

The method for deriving the summary conditions of a plan p is recursive. First, summary information must be derived for each of p's subplans, and then the following procedure derives p's summary conditions from those of its subplans and its own sets of conditions. This procedure only apply to plans whose expansion is finite and which have the downward solution property. This is the property where every *or* plan in the hierarchy can be refined successfully through one or more of its subplans.

Summary conditions for primitives and non-primitives

- First, for each literal ℓ in $pre(p)$, $in(p)$, and $post(p)$, add a condition c with literal ℓ to the respective set of summary conditions for plan p. $existence(c)$ is *must*, and $timing(c)$ is *first*, *always*, or *last* if ℓ is a pre-, in-, or postcondition respectively.

Summary [pre, post] conditions for *and* plan

- Add a condition c to the summary [pre, post] conditions of *and* plan p for each summary [pre, post] condition c' of p's subplans that is not [*must-achieved*, *must-undone*][5] by another of p's subplans, setting $\ell(c) = \ell(c')$.[6]
- Set $existence(c) = must$ if $\ell(c)$ is a [pre, post] condition of p or is the literal of a *must* summary [pre, post]

[5] See (Clement & Durfee 1999a) and the proof ending this section about how to determine *must-achieved*, *may-achieved*, *must-undone*, and *may-undone*.

[6] To resolve ambiguity with set membership, we say that any two summary conditions, c and c' are equal if $\ell(c) = \ell(c')$ and they belong to the same set of summary conditions for some plan.

condition in a subplan of p that is not [*may-achieved*, *may-undone*] by any other subplans. Otherwise, set $existence(c) = may$.
- Set $timing(c) = [first, last]$ if $\ell(c)$ is a [pre, post] condition of p or the literal of a [*first*, *last*] summary [pre, post] condition of a [least, greatest] temporally ordered subplan (i.e. no others are constrained by $order(p)$ to [begin before, end after] it). Otherwise, set $timing(c) = sometimes$.

Summary [pre, post] conditions for *or* plan

- Add a condition c to the summary [pre, post] conditions of *or* plan p for each summary [pre, post] condition c' in p's subplans, setting $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is a [pre, post] condition of p or a *must* summary [pre, post] condition of all of p's subplans. Otherwise, set $existence(c) = may$.
- Set $timing(c) = [first, last]$ if $\ell(c)$ is a [pre, post] condition of p or the literal of a [*first*, *last*] summary [pre, post] condition in a subplan. Otherwise, set $timing(c) = sometimes$.

Summary inconditions for *and* plan

- Add a condition c to the summary inconditions of *and* plan p for each c' in C defined as the set of summary inconditions of p's subplans unioned with the set of summary preconditions of the subplans that are not *first* in a least temporally ordered subplan and with the set of summary postconditions of the subplans that are not *last* in a greatest temporally ordered subplan, and set $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is an incondition of p or a literal of a *must* summary condition $c' \in C$, as defined above. Otherwise, set $existence(c) = may$.
- Set $timing(c) = always$ if $\ell(c)$ is an incondition of p or a literal in an *always* summary incondition in every subplan of p. Otherwise, set $timing(c) = sometimes$.

Summary inconditions for *or* plan

- Add a condition c to the summary inconditions of *or* plan p for for each summary incondition c' in p's subplans, setting $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is an incondition of p or a *must* summary incondition of all of p's subplans. Otherwise, set $existence(c) = may$.
- Set $timing(c) = always$ if $\ell(c)$ is an incondition of p or an *always* summary incondition of all of p's subplans. Otherwise, set $timing(c) = sometimes$.

Consider deriving the summary conditions of m_2 from its two primitive subplans (as introduced in the section on CHiPs). Suppose p_1 is the primitive subplan for moving agent A from (1,1) to (1,2), and p_2 moves A from (1,2) to (1,3). First, the summary conditions of the primitives must be derived. These are simply the conditions already defined for the primitives according to the first step of the procedure. m_2 has no conditions defined for itself, so all will come from p_1 and p_2. Since m_2 is an *and* plan, its only summary precondition is $At(A,1,1)$ from p_1 because p_1 *must achieve* p_2's only precondition $At(A,1,2)$. $At(A,1,1)$ is a *must* summary condition because it is a *must* summary precondition in p_1, and no other subplan (p_2) *may achieve* $At(A,1,1)$. $At(A,1,1)$ is also *first* because it is a *first* summary precondition of p_1, and p_1 precedes p_2.

The procedure above ensures that external conditions are captured by summary conditions and *must*, *always*, *first*, and *last* have their intended meanings. The actual proof of these properties is all-inclusive since the truth of each property depends on those of others. However, we give a proof of one (assuming the others) to illustrate how we verify these properties using the language developed in this paper. The full proof is given in an extended report (Clement & Durfee 1999a). These results ease the proofs of soundness and completeness for inference rules determining how CHiPs can definitely or potentially interact so that good planning and coordination decisions can be made at various levels within and among plan hierarchies.

Theorem The set of external preconditions for a plan is equivalent to the set of all literals in the plan's summary preconditions.

Proof by induction over the maximum subplan depth. The base case is a primitive plan p (subplan depth zero). The summary preconditions include a condition for every precondition of p, which must be an external precondition of p, so this case is satisfied. Assume that the theorem is true for all plans of maximum depth $\leq k$. Any plan p of maximum depth $k+1$ must have subplans with maximum depths $\leq k$. It is not difficult to show that the external preconditions of p must be the preconditions of p and those external preconditions (call them pre_x) of p's subplans that are not *must-achieved* (achieved in all executions) by another subplan of p.

By the inductive hypothesis, the external conditions of the subplans are captured in their summary conditions, and the *existence* and *timing* information together with the *order* of p's subplans can be used to determine whether some external precondition of a subplan is *must-achieved*. Table 1 shows this by describing for all cases the constraints on $order(p)$ where $\ell(c')$ of p' is *must-achieved* by p''. If we did not assume the downward solution property, then we would additionally need to make sure that no other plan could clobber $\ell(c')$ after p'' asserts the condition. Hence, the external preconditions in pre_x that are not *must-achieved* are exactly those determined in the rule for determining the summary preconditions of an *and* plan. Therefore, the external conditions of p are exactly those described as the summary preconditions of p in the procedure described above. □

Complexity

The procedure for deriving summary conditions works by basically propagating the conditions from the primitives up the hierarchy to the most abstract plans. Because the conditions of any non-primitive plan depend only on those of its immediate subplans, deriving summary conditions can be done quickly. Given that an agent has an instantiated plan hierarchy with n non-primitive plans, each of which have b subplans and c conditions in each of their summary pre-, in-, and

$c'' \in post_{sum}(p'')$		$c' \in pre_{sum}(p')$		p'' *must-achieve* c' $\forall e' \in \mathcal{E}(p'), e'' \in \mathcal{E}(p'')$
must	*last*	*must*	*first*	
F	?	?	?	*false*
T	?	?	?	$t_f(e'') \leq t_s(e')$
$c'' \in in_{sum}(p'')$				
must	*always*	*must*	*first*	
F	?	?	?	*false*
?	F	?	?	*false*
T	T	?	F	$t_s(e'') \leq t_s(e') \wedge$ $t_f(e') \leq t_f(e'')$
		?	T	$t_s(e'') < t_s(e') < t_f(e'')$

Table 1: Ordering constraints necessary for subplan p'' to *must-achieve* $c' \in pre_{sum}$ of subplan p'. "?" means that the constraints hold for both truth values. *false* means that there are no ordering constraints guaranteeing that p' is achieved by p''.

postconditions, deriving the summary conditions of the non-primitive plans can be bounded by $O(nb^2c^2)$ operations. This comes from the worst case in which all plans are *and* plans requiring the procedure to test each of c conditions in each of b subplans to see if they are achieved/clobbered by any those same conditions in any of the same subplans for each of the n *and* plans. However, $n = O(b^d)$ for hierarchies of depth d, so the complexity of the procedure for deriving summary conditions is more simply $O(n(log^2 n)c^2)$.

Soundness and Completeness of Determining Temporal Relations

With the properties of summary information proven, we can safely reason about the interactions of plans without information about their subplans. Based on the *existence* and *timing* information carried by summary conditions, we can determine what relations can or might hold between CHiPs without searching their subplans. In many coordination tasks, if it could be determined that certain temporal relations can hold among plans no matter how they are decomposed ($CanAnyWay$) or that certain relations cannot hold for any decomposition ($\neg MightSomeWay$), then coordination decisions can be made at abstract levels without entering a potentially costly search for valid plan merges. Here we prove the soundness and completeness of rules determining $CanAnyWay$ and $MightSomeWay$ relations based on summary information. The use of these rules is illustrated in the Introduction and explored further in (Clement & Durfee 1999b). For convenience, we will abbreviate *Can* to *C*, *Any* to *A*, *Way* to *W*, and so on.

Informally, [$CAW(rel, p_{sum}, q_{sum})$, $MSW(rel, p_{sum}, q_{sum})$] says that the temporal relation *rel* [*can*, *might*] hold for any CHiPs p and q whose summary information is p_{sum} and q_{sum} for [*any way*, *some way*] that p and q may be executed. We formally describe these relations by explaining what soundness and completeness mean for rules to determine them.

Let us define $AW(P, s_I)$ to mean that in any history h with initial conditions s_I and where $E(h)$ includes an execution of each plan in set P as well as its subexecutions, all executions succeed. Also, let [$AW(rel, p, q)$, $SW(rel, p, q)$] be true iff for [any, some] history h where $AW(\{p\}, s_I(h))$ and $AW(\{q\}, s_I(h))$ are true, and $E(h)$ includes executions of p and q and their subexecutions satisfying relation *rel*, all executions succeed. So, a rule for determining [$CAW(rel, p_{sum}, q_{sum})$, $MSW(rel, p_{sum}, q_{sum})$] is sound if whenever the rule returns *true*, [$AW(rel, p, q)$, $SW(rel, p, q)$] is true for [all pairs, some pair] of plans whose summary information is p_{sum} and q_{sum}. The rule is complete if whenever [$AW(rel, p, q)$, $SW(rel, p, q)$] is true for [all, some pair of] plans p and q with summary conditions p_{sum} and q_{sum}, the rule also returns *true*.

Now we state the rules for determining *overlaps*. $CAW(\mathrm{o}, p_{sum}, q_{sum})$ returns *true* iff there is no c_p and c_q such that $\ell(c_p) \Leftrightarrow \neg\ell(c_q)$ and either $c_p \in in_{sum}(p)$ and $c_q \in pre_{sum}(q) \cup in_{sum}(q)$ or $c_p \in post_{sum}(p)$ and $c_q \in in_{sum}(q)$. $MSW(\mathrm{o}, p_{sum}, q_{sum})$ returns *true* iff there is no c_p and c_q such that $\ell(c_p) \Leftrightarrow \neg\ell(c_q)$; $must(c_p)$ and $must(c_q)$ is true; the *timing* of c_p and c_q is either *first*, *always*, or *last*; and either $c_p \in in_{sum}(p)$ and $c_q \in pre_{sum}(q) \cup in_{sum}(q)$ or $c_p \in post_{sum}(p)$ and $c_q \in in_{sum}(q)$. Note that for n conditions in each of the pre-, in-, and postconditions of p and q, in the worst case each condition in one plan must be compared with each in another plan, so the complexity of determining CAW and MSW is $O(n^2)$. Now we will show that the above rules are both sound and complete. The proofs of rules for other temporal relations are no more difficult than this one and can be done constructively in the same style.

Theorem $CAW(\mathrm{o}, p_{sum}, q_{sum})$ and $MSW(\mathrm{o}, p_{sum}, q_{sum})$ are sound and complete.

Proof Since we assume $AW(\{p\}, s_I)$ and $AW(\{q\}, s_I)$, the only way executions of p or q could fail is if a condition in one is clobbered by the other. However, a condition is clobbered only by the assertion of the negation of its literal. Thus, the presence of conditions involving the propositional variable v cannot clobber or achieve conditions involving the propositional variable v' if $v \neq v'$. In addition, all cases of execution failure can be broken down into inconditions of some execution e clobbering conditions that must be met in the interval $(t_s(e), t_f(e))$ and postconditions of some execution e clobbering conditions at or after $t_f(e)$. With this established it is not difficult to show that the only pairs of interacting sets where clobbering occurs for the *overlaps* relation include $(in_{sum}(p), pre_{sum}(q))$, $(in_{sum}(p), in_{sum}(q))$, and $(post_{sum}(p), in_{sum}(q))$. And, because the clobbering of a single literal causes execution failure, we can describe all histories in terms of the presence and absence of summary conditions based on a single propositional variable.

This is done for the *overlaps* relation in Table 2. Here, we give all instances of these interacting sets and claim that the listed truth values for CAW and MSW

	A		B		$A = in_{sum}(p)$ $B = pre_{sum}(q)$		$A = in_{sum}(p)$ $B = in_{sum}(q)$		$A = post_{sum}(p)$ $B = in_{sum}(q)$	
					CAW	MSW	CAW	MSW	CAW	MSW
1	-		?		T	T	T	T	T	T
	?		-		T	T	T	T	T	T
2	ℓ		ℓ		T	T	T	T	T	T
3	ℓ		$\neg\ell$							
	m	a/l	m	f/a						
a	T	T	T	T	F	F	F	F	F	F
b	F	?	?	?	F	T	F	T	F	T
c	?	F	?	?	F	T	F	T	F	T
d	?	?	F	?	F	T	F	T	F	T
e	?	?	?	F	F	T	F	T	F	T
4	$\ell, \neg\ell$		ℓ		F	T	F	T	F	T
5	ℓ		$\ell, \neg\ell$		F	T	F	T	F	T
6	$\ell, \neg\ell$		$\ell, \neg\ell$		F	T	F	T	F	T

Table 2: Truth values of $CanAnyWay(\mathrm{o}, p, q)$ and $MightSomeWay(\mathrm{o}, p, q)$ for all interactions of conditions. ℓ and $\neg\ell$ are literals of summary conditions in sets A and B. The condition is *must* if m = T, *first* for f = T, *last* for l = T, and *always* for a = T.

are correct for the space of plan pairs (p, q) that elicit the instances represented by each row. The literal ℓ in the first two columns represents any literal that appears in any condition of the set in whose column it appears. For example, row 4 for the interaction of $(in_{sum}(p), pre_{sum}(q))$ is interpreted as the case where there is a literal that appears in both sets, and its negation also appears in just $in_{sum}(p)$. [F, T] in the [CAW, MSW] column means that [*not all* histories, there is at least one history] with only the executions and subexecutions of [any, some] pair of plans whose summary conditions have literals appearing this way are such that all executions succeed. Thus, [CAW(o, p_{sum}, q_{sum}), MSW(o, p_{sum}, q_{sum})] is true iff there are no summary conditions matching cases in the table where there is an F entry in a [CAW, MSW] column.

The validity of most entries in Table 2 is simple to verify. In row 1 either there is no condition to be clobbered or no condition to clobber it, so all executions must succeed. In row 3 unless the conditions are all *must* and either *first*, *always*, or *last* (row 3a), there is always a history containing the executions of some plans where the two conflicting conditions are not required to be met at the same time, and MSW is true. Because rules defined for CAW(o, p_{sum}, q_{sum}) and MSW(o, p_{sum}, q_{sum}) return *true* whenever the table has a T entry, they are complete. And because the table has a T entry for every case in which the rules return *true*, the rules are sound. □

Conclusions and Future Work

Coordination and planning for hierarchical plans often involve the refinement of abstract plans into more detail. However, the cost of backtracking or making irreversible commitments makes it critical that the specification of abstract plan operators be rich enough to anticipate and avoid costly planning/execution decisions. We have addressed this directly by offering a formalism for describing concurrent hierarchical plan execution and methods for deriving summary conditions and determining legal plan interactions in a sound and complete fashion. This provides a foundation upon which provably correct coordination and planning mechanisms can be built. In prior work that motivates and validates this formalism, we describe a general algorithm for merging hierarchical plans using summary information, a specific implementation, and a preliminary evaluation of the approach (Clement & Durfee 1999b). Future work includes relaxing assumptions, such as the downward solution property, investigating other types of plan summary information, and constructing sound and complete algorithms for concurrent hierarchical planning and for interleaving planning, plan coordination, and plan execution.

References

Allen, J. F., and Koomen, J. A. 1983. Planning using a temporal world model. In *Proc. IJCAI*, 741–747.

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Clement, B., and Durfee, E. 1999a. Theory for coordinating concurrent hierarchical planning agents. http://www.eecs.umich.edu/~bradc/papers/aaai99.

Clement, B., and Durfee, E. 1999b. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. Intl. Conf. Autonomous Agents.*

Corkill, D. 1979. Heirarchical planning in a distributed environment. In *Proc. IJCAI*, 168–175.

Ephrati, E., and Rosenschein, J. 1994. Divide and conquer in multi-agent planning. In *Proc. AAAI*, 375–380.

Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about knowledge.* MIT Press.

Firby, J. 1989. *Adaptive Execution in Complex Dynamic Domains.* Ph.D. Dissertation, Yale University.

Georgeff, M. P., and Lansky, A. 1986. Procedural knowledge. *Proc. IEEE* 74(10):1383–1398.

Georgeff, M. P. 1983. Communication and interaction in multiagent planning. In *Proc. AAAI*, 125–129.

Georgeff, M. P. 1984. A theory of action for multiagent planning. In *Proc. AAAI*, 121–125.

Lansky, A. 1990. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, 115–125.

Sacerdoti, E. D. 1977. *A structure for plans and behavior.* Elsevier-North Holland.

Tate, A. 1977. Generating project networks. In *Proc. IJCAI*, 888–893.

Tsuneto, R.; Hendler, J.; and Nau, D. 1998. Analyzing external conditions to improve the efficiency of htn planning. In *Proc. AAAI*, 913–920.

Vilain, and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proc. AAAI*, 377–382.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Appendix D: Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information

Bradley J. Clement and Edmund H. Durfee

Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110, USA
+1-734-764-2138
{bradc, durfee}@umich.edu

Abstract. Recent research has provided methods for coordinating the individually formed concurrent hierarchical plans (CHiPs) of a group of agents in a shared environment. A reasonable criticism of this technique is that the summary information can grow exponentially as it is propagated up a plan hierarchy. This paper analyzes the complexity of the coordination problem to show that in spite of this exponential growth, coordinating CHiPs at higher levels is still exponentially cheaper than at lower levels. In addition, this paper offers heuristics, including "fewest threats first" (FTF) and "expand most threats first" (EMTF), that take advantage of summary information to smartly direct the search for a global plan. Experiments show that for a particular domain these heuristics greatly improve the search for the optimal global plan compared to a "fewest alternatives first" (FAF) heuristic that has been successful in Hierarchical Task Network (HTN) Planning.

1 Introduction

In a shared environment with limited resources, agents may have enough information about the environment to individually plan courses of action but may not be able to anticipate how the actions of others will interfere with accomplishing their goals. Prior techniques have enabled such agents to cooperatively seek merges of individual plans that will accomplish all of their goals if possible [7]. This is done by identifying conflicts and adding synchronization actions to the plans to avoid conflicts. Agents can also interleave planning and merging, such that they propose next-step extensions to their current plans and reconcile conflicts before considering extensions for subsequent steps. By formulating extensions in terms of constraints rather than specific actions, a "least commitment" policy can be retained [5]. In addition, recent research has provided these agents with tools to coordinate their hierarchical plans resulting in more flexible abstract solutions that allow the agents to choose refinements of their actions during execution that can withstand some amount of failure and uncertainty [3]. In addition to adding ordering constraints, agents may need to eliminate choices of subplans for accomplishing subgoals. In order to reason about abstract plans to identify and resolve conflicts, information about how the abstract plans must or may be refined into lower level actions must be available. This information can be summarized from the conditions of subplans in its potential refinements.

It was previously shown that using this strategy to find abstract solutions to the coordination problem can improve the overall performance of coordinating and executing plans [3]. As depicted in Figure 1, coordination is cheaper at higher levels in the hierarchy because there are fewer plan steps to reason about. Although anecdotal evidence was given to show this, in this paper we reinforce the result with a more rigorous complexity analysis. At lower levels in the hierarchy, however, more detailed solutions of potential greater quality can be found, but only after greater coordination effort. Depending on how costly computation time is compared to the cost of executing the coordinated plans, coordinating at levels in between the top and bottom could likely result in better overall performance. On the other hand, only coordinating at the lowest level can guarantee finding the optimal solution.

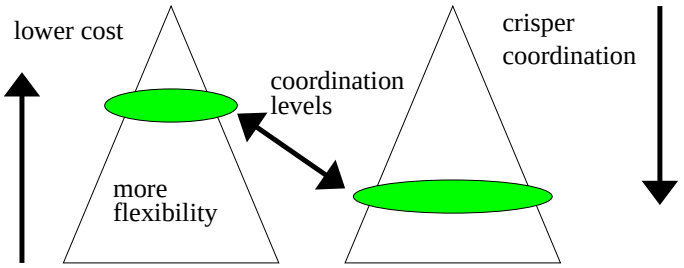


Fig. 1. Hierarchical plan coordination at multiple levels.

If the goal is to find the optimal solution, a reasonable criticism might be that using summary information to reason at abstract levels will be more costly than just coordinating at the lowest level of primitive actions because of the overhead of deriving and using summary information. The experimental results given here contradict this criticism and show how reasoning about plans at abstract levels can better focus the search to much more quickly find detailed solutions at the level of primitive actions.

This paper makes the following contributions:

- complexity analysis showing that finding global plans at higher levels can be exponentially less expensive than at lower levels;
- search techniques and heuristics, including Fewest Threats First (FTF) and Expand Most Threats First (EMTF), that take advantage of summary information;
- a description of a search algorithm that uses these heuristics for coordinating concurrent hierarchical plans; and
- preliminary experiments showing how these heurisitics can greatly save computation time in finding the optimal plan compared to a Fewest Alternatives First (FAF) heuristic [4] that has been successful in Hierarchical Task Network (HTN) Planning [8].

In addition, of potential interest to the planning community, we prove that resolving threats among a set of unordered STRIPS operators is NP-complete. This result is necessary our complexity analysis.

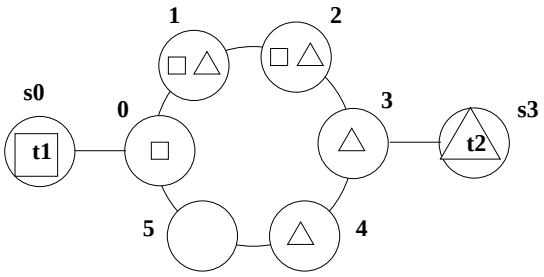


Fig. 2. Transports $t1$ and $t2$ must pick up square and triangle evacuees respectively.

Reasoning about abstract plans with conditions of lower-level subplans has also been used to efficiently guide the search through hierarchical plan spaces (HTN planning) for single agents [9]. Their technique computes the *external conditions* of abstract plans, which are the preconditions required external to the abstract plans in order for them to be executed successfully. We redefine these as *external preconditions* and additionally employ *external postconditions*, the effects seen external to an abstract plan. Since the coordination problem requires reasoning about the concurrent execution of actions, we also derive *summary inconditions*, the intermediate, or internal, conditions that must or may be required to hold during an abstract plan step for the execution to be successful. We have detailed a procedure for deriving these summary conditions, proofs of their properties, and sound and complete mechanisms for determining legal interactions of abstract plans based on summary conditions elsewhere in [2].

1.1 A Simple Example

This example illustrates how agents can coordinate their actions using summary information to guide the search for a global plan that resolves conflicts and optimizes the total completion time of the agents' plans. In a non-combative evacuation operation (NEO) domain, transport agents are responsible for visiting certain locations along restricted routes to pick up evacuees and bring them back to safety points. To avoid the risk of oncoming danger (from a typhoon or an enemy attack), the transports need to coordinate in order to avoid collisions along the single lane routes and must accomplish their goals as quickly as possible.

Suppose there are two transport agents, t 1 and $t2$, located at safety points $s0$ and $s3$ respectively, and they are responsible for visiting the locations 0-2 and 1-4 respectively as shown in Figure 2. Because there is overlap in the locations they must visit, they must synchronize their individual plans in order to avoid a collision. The hierarchical plan of $t1$ at the highest level is to evacuate the locations for which it is responsible. This decomposes into a primitive action of moving to location 0 on the ring and then to traverse the ring. It can choose to adopt a plan to travel in one direction around the ring without switching directions, or it can choose to switch directions once. $t1$ can then choose to either go clockwise or counterclockwise and, if switching, can choose to switch directions at any location and travel to the farthest location it needs to visit from

where it switched. Once it has visited all the locations, it continues around until it can go to the first safety point it finds. $t2$ has a similar plan.

Now let us say $t1$ collects summary information about $t2$'s plan and attempts to coordinate it with its plan. Looking just at the highest level, $t1$ can determine that if it finishes its plan before $t2$ even begins execution, then there will be no conflicts since the external postconditions of its $evacuate$ plan reveal that none of the routes are being traversed. $t1$ then tells $t2$ to add a plan step to the beginning of its plan to $wait$ for $t1$'s signal, and $t1$ can append a $signal$ subplan to the end of its plan. However, this coordinated global plan is inefficient since there is no parallel action—the length ranges from 12 to 26 steps depending on how the agents decompose their plans during execution. If the agents wish to get more concurrency, then they must expand the top-level plans into more detailed plans and resolve conflicts there. At a mid-level expansion where both agents move clockwise without switching directions, the algorithm finds a solution with a length of only eight steps. Now the search algorithm can eliminate the need to resolve threats for any global plan whose length can be no shorter than eight. To find the optimal solution, the agents must almost completely expand their hierarchies. This is a plan of length seven where $t1$ moves clockwise until it reaches location $s3$, and $t2$ starts out clockwise, switches at location 4, and then winds up at $s0$.

1.2 Overview

In the next section, we describe how concurrent hierarchical plans can be coordinated using summary information. Then we explain why it is easier to compute abstract solutions at higher levels than at lower levels with a complexity analysis of the coordination algorithm. Next we show experimental results verifying that summary information can greatly improve the search for the optimal global plan even when it exists at the lowest level of primitive actions.

2 Top-Down Coordination of Concurrent Hierarchical Plans

Our approach to coordinating concurrent hierarchical plans (CHiPs) is to first try to coordinate the plans at the top-level of the hierarchies, then consider their subplans, and iteratively expand selected subplans until a "feasible" solution is found. A general algorithm for doing this is described in [3]. Here we briefly explain the basic mechanisms for deriving and using summary information and then describe a specific algorithm we use to evaluate the effectiveness of coordinating using summary information. All terms and mechanisms mentioned here are formalized in [2].

2.1 CHiPs

As described here, hierarchical plans are non-primitive plans that each have their own sets of conditions, a set of subplans, and a set of ordering constraints over the subplans. These ordering constraints can be conjunctions of temporal interval relations [1] or point relations over endpoints of plan execution time intervals. A primitive plan is only different in that it has an empty set of subplans. In the style of STRIPS planning

operators [6], each of these plans has sets of preconditions and effects. [1] However, since we necessarily worry about agents performing tasks in parallel, we also associate a set of inconditions with each plan so that threats during the execution of a task can be represented.

An agent's plan library is a set of CHiPs, any of which could be part of the agent's current plan, and each plan in the hierarchy is either a primitive plan, an and plan, or an or plan. An and plan decomposes into a set of plans that each must be accomplished according to specified temporal constraints. An or plan decomposes into a set of plans of which only one must be accomplished. So, for the example given in Section 1.1, there is an or plan that would have subplans for traveling clockwise or counterclockwise, and there are and plans for chaining primitive level movements between locations to get a transport around the ring.

2.2 Plan Summary Information

We derive summary conditions for CHiPs by propagating the conditions from the primitive level up the hierarchy. The procedure is quick ($O(n^2c^2)$ for n plans in the hierarchy each with c conditions) because the summary conditions of a plan are derived only from its own conditions and the summary conditions of its immediate subplans. As mentioned in the Section 1, summary preconditions, inconditions, and postconditions are computed for each plan to represent the external preconditions, internal conditions, and external postconditions respectively. Modal information about whether these conditions $must$ or may hold and whether they must hold throughout the plan's execution ($always$ or $sometimes$) is kept to reason about whether certain plan interactions must or may occur.

2.3 Temporal Interactions of CHiPs

In conventional planning, we often speak of *clobbering* and *achieving* preconditions of plans [10]. With CHiPs these notions are slightly different since inconditions can clobber and be clobbered. We use these concepts to determine whether a summary precondition of a plan should be a summary condition of its parent. A summary precondition is an *external* precondition of its subplans, and what makes the precondition external is that it is not *achieved* by another subplan—it needs to be met outside the scope of the parent plan. A summary postcondition is external because it is a net effect of the execution of the subplans. Thus, we need to also determine when a postcondition is *undone* by another subplan since a postcondition is not external if it is undone.

Determining these relationships helps us derive summary information, but it also helps identify threats across the plan hierarchies of the agents. For example, plan p of one agent cannot clobber a condition c of plan q of another agent if there is another plan r ordered between p and q that achieves c for q. However, if plan r only may achieve c because c is a may postcondition of r, then p threatens q. Reasoning about these kinds of interactions, we can determine that a set of temporal relations can hold among plans no matter how they are decomposed ($CanAnyWay$) or that certain relations cannot

[1] These are not summary conditions.

hold for any decomposition ($\neg MightSomeWay$). As the procedure for determining these relations is similar to propagating summary information, its complexity is also $O(n^2c^2)$ for n plans with c conditions each [2].

2.4 Top-Down Search Algorithm

Since we can determine whether abstract plans $CanAnyWay$ or $MightSomeWay$ be executed successfully under particular ordering constraints, we can integrate this into an algorithm that smartly searches for a consistent global plan for a group of agents. The particular algorithm we describe here is complete, and returns the optimal global plan if it exists. The search starts out with the top-level plan of each agent which together represent the global plan. It tries to find a solution at this level and then expands the hierarchies deeper and deeper until the optimal solution is found or the search space has been exhausted. A pseudocode description of the algorithm is given below.

A state of the search is a partially elaborated global plan that we represent as a set of and plans (one for each agent), a set of temporal constraints, and a set of blocked plans. The subplans of the and plans are the leaves of the partially expanded hierarchies of the agents. The set of temporal constraints includes synchronization constraints added during the search in addition to those dictated by the agents' individual hierarchical plans. Blocked subplans keep track of pruned or subplans.

The operators of the search are expanding non-primitive plans, blocking or subplans, and adding temporal constraints on pairs of plans. When a plan is expanded, it is replaced by its subplans, and the ordering information is updated in the global plan. Or plans are only replaced by a subplan when all other subplans are blocked.

Blocking an or subplan can be effective in resolving a constraint in which the other or subplans are not involved. This can lead to least commitment abstract solutions that leave the agents flexibility in selecting among multiple applicable subplans. Another approach is to select subplans (effectively blocking the others) to investigate choices that are given greater preference or are more likely to resolve conflicts.

In the pseudocode below, the coordinating agent collects summary information about the other agents' plans as it decomposes them. The $queue$ keeps track of expanded search states. If the $CanAnyWay$ relation holds for the search state, the Dominates function determines if the current solutions are better for every agent than the solution represented by the current search state and keeps it if the solution is not dominated. If $MightSomeWay$ is false, then the search space represented by the current search state can be pruned; otherwise, the operators mentioned above are applied to generate new search states. Nondeterministic "Choose" functions determine how these operators are applied. Our implementation uses heuristics specified in Section 2.5 to determine what choices are made. When a plan is expanded or selected, the ordering constraints for that plan must be updated for the subplans that replace it. The Update-Order function accomplishes this.

Hierarchical Plan Coordination Algorithm

```
plans = ∅
for each agent a_i
   p_i = get summary information for top-level plan
```

```
   plans = plans ∪ {p_i}
end for
queue = {(plans, ∅, ∅)}
solutions = ∅
loop
   if queue == ∅
      return solutions
   end if
   (plans, order, blocked) = Pop(queue)
   if CanAnyWay(initial_state, plans, order, blocked)
      solution = (plans, order, blocked)
      if Dominates(solutions, solution) == false
         solutions = solutions ∪ {solution}
      end if
   end if
   if MightSomeWay(initial_state, plans, order, blocked)
      operator = Choose({expand, select, block, constrain})
      if operator == expand
         plan = ChooseAndPlan(plans)
         if Exists(plan)
            plan.subplans = get summary information for subplans of plan
            plans = plans ∪ plan.subplans - plan
            UpdateOrder(order, plan, plan.subplans, plan.order)
         end if
      end if
      if operator == select
         plan = ChooseOrPlan(plans)
         if Exists(plan)
            plan.subplans = get summary information for subplans of plan
            for each subplan ∈ plan.subplans
               newblocked = blocked ∪ plan.subplans - {subplan}
               newplans = plans ∪ {subplan} - plan
               neworder = order
               UpdateOrder(neworder, plan, {subplan}, ∅)
               InsertStateInQueue(queue, newplans, neworder, newblocked)
            end for
         end if
      end if
      if operator == block
         plan = ChooseOrPlan(plans)
         if Exists(plan)
            plan.subplans = get summary information for subplans of plan
            for each subplan ∈ plan.subplans where subplan ∉ blocked
               newblocked = blocked ∪ subplan
               neworder = order
               if ∃! subplan' ∈ plan.subplans, subplan' ∉ blocked
                  newplans = plans ∪ {subplan'} - plan
                  UpdateOrder(neworder, plan, {subplan'}, ∅)
               else
                  newplans = plans
```

```
                end if
                InsertStateInQueue(queue, newplans, neworder, newblocked)
             end for
          end if
       end if
       if operator == constrain
          plan = ChoosePlan(plans)
          plan' = ChoosePlan(plans - {plan})
          constraint = ChooseConstraint({Start, End} × {<, ≤, =, ≥, >} × {Start, End})
          neworder = order ∪ constraint
          if Consistent(neworder)
             InsertStateInQueue(queue, plans, neworder, blocked)
          end if
       end if
    end if
 end loop
```

Adding temporal constraints should only generate new search nodes when the ordering is consistent with the other global and local constraints. In essence, this operator performs the work of merging non-hierarchical plans since it is used to find a synchronization of the individual agents' plans that are one level deep. In the pseudocode above, the ChooseConstraint function nondeterministically investigates all orderings, and inconsistent ordering constaints are pruned. However, in our implementation, we only investigate legal ordering constraints that resolve threats that are identified by algorithms determining must/may achieves and clobbers relations among CHiPs. In our experiments, we separated the search for synchronizations from the expansion and selection of subplans. An outer search was used to explore the space of plans at different levels of abstraction. For each state in the outer search, an inner search explores the space of plan merges by resolving threats with ordering constraints.

⟨ Paragraph here arguing soundness and completeness. ⟩

2.5 Heuristics Using Summary Information

As discussed in [3], summary information is valuable for finding coordinated plans at abstract levels. However, this information can also be valuable in directing the search to avoid branches in the search space that lead to inconsistent or suboptimal global plans. Inconsistent global plans can be pruned away at the abstract level by doing a quick check to see if $MightSomeWay$ is false. In terms of the number of states expanded during the search, employing this technique will always do at least as well as not using it. Another strategy that is employed is to first expand plans involved in the most threats. For the sake of completeness, the order of plan expansions does not matter as long as they are all expanded at some point when the search trail cannot be pruned. But, employing the "expand on most threats first" (EMTF) heuristic aims at driving the search down through the hierarchy to find the subplan(s) causing conflicts with others so that they can be resolved more quickly. This is similar to a most-constrained variable heuristic often employed in constraint satisfaction problems. Another heuristic used in parallel in our experiments is "fewest threats first" (FTF). Here the search orders nodes

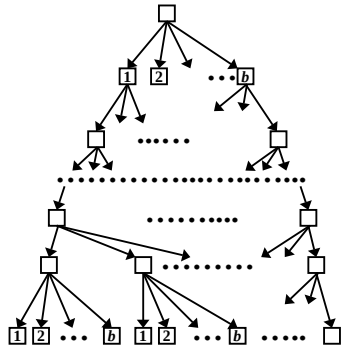
in the outer search queue by ascending numbers of threats to resolve. By trying to resolve the threats of global plans with fewer conflicts, it is hoped that solutions can be found more quickly. So, EMTF is a heurisitic ordering plans to expand, and FTF orders subplan choices and, thus, search states to investigate. In addition, in trying to find optimal solutions in the style of a branch-and-bound search, we use the cost of abstract solutions to prune away branches of the search space whose minimum cost is greater than the maximum cost of the current best solution. This technique can be used without summary information, but then only solutions at the primitive level can be used to prune the search space. Again, pruning abstract plans can only help improve the search. We report experimental results in Section 4 that show that these techniques and heuristics can greatly improve coordination performance.

3 Complexity

In [3], anecdotal evidence was given to show that coordinating at higher levels of abstraction is less costly because there are fewer plan steps. But, even though there are fewer plans at higher levels, those plans have greater numbers of summary conditions to reason about because they are collected from the much greater set of plans below. Here we argue that even in the worst case where summary conditions increase exponentially up the hierarchy, finding solutions at abstract levels is expected to be exponentially cheaper than at lower levels.

The procedure for deriving summary conditions works by basically propagating the conditions from the primitives up the hierarchy to the most abstract plans. Because the conditions of any non-primitive plan depend only on those of its immediate subplans, deriving summary conditions can be done quickly. In [2], it was reported that the complexity of this is $O(n(log^2 n)c^2)$ for n non-primitive plans with c conditions in each plan's *summary* pre-, in-, and postconditions. This, however, does not tell us how the complexity grows as a result of summary conditions accumulating in greater and greater sets as they are propagated up the hierarchy. If c' is the greatest number of literals in any plan's pre-, in-, and postconditions, then the complexity is $O(n^2 c'^2)$. Here, the worst case is when all plans are and plans, and the conditions of each plan are completely different than those of any other plan. In this way, the maximum number of conditions are propagated up the hierarchy and all of the expanded plans must be synchronized to avoid conflicts. Consider a global hierarchy with n total plans, b subplans for each non-primitive plan, and depth d.[2] At each level, the procedure tests each condition in each summary condition set of the b subplans of each plan at that level to see if they are achieved/clobbered/undone by any other subplan attempting to assert that condition. Thus, a constant number of operations must be performed when comparing each condition in each subplan with every other condition in every other subplan resulting in $O(b^2c^2)$ operations for each plan with b subplans each having $O(c)$ summary conditions. So, as shown in Figure 3, at the next-to-bottom depth level $d-1$, each of the b^{d-1} plans has b primitive subplans each with $O(c')$ conditions. Thus, $O(b^2c'^2)$ operations are performed for each of the b^{d-1} plans for a total of $O(b^{d-1}b^2c'^2)$ op-

[2] We consider the root at depth level 0 and the leaves at level d.



level	#plans	#conds / plan	#operations to derive summ. info.	#test operations / solution candidate	solution space
0	1	$O(b^d c')$	$O(b^2(b^{d-1}c')^2) = O(b^{2d}c'^2)$	$O(1)$	1
1	b	$O(b^{d-1}c')$	$O(bb^2(b^{d-2}c')^2) = O(b^{2d-1}c'^2)$	$O(b^2(b^{(d-1)}c')^2) = O(b^{2d}c'^2)$	$O(b!)$
2	b^2	$O(b^{d-2}c')$	$O(b^2b^2(b^{d-3}c')^2) = O(b^{2d-2}c'^2)$	$O(b^4(b^{(d-2)}c')^2) = O(b^{2d}c'^2)$	$O(b^2!)$
d-2	b^{d-2}	$O(b^2c')$	$O(b^{d-2}b^2(bc')^2) = O(b^{d+2}c'^2)$	$O(b^{2(d-2)}(b^2c')^2) = O(b^{2d}c'^2)$	$O(b^{d-2}!)$
d-1	b^{d-1}	$3c'+b3c' = O(bc')$	$O(b^{d-1}b^2c'^2) = O(b^{d+1}c'^2)$	$O(b^{2(d-1)}(bc')^2) = O(b^{2d}c'^2)$	$O(b^{d-1}!)$
d	b^d	$3c'$	$O(1)$	$O(b^{2d}c'^2)$	$O(b^d!)$

Fig. 3. The table gives the number of plans and summary conditions for each plan at some level of expansion of a global plan hierarchy (with branching factor b) where each plan has c' conditions in each set of pre-, in-, and postconditions. The number of operations to derive summary information for all of the plans at a particular depth level is the product of the number of plans at that level, the square of the number of subplans per plan, and the square of the number of conditions per subplan. The number of operations to check if a candidate expansion under particular ordering constraints is a solution is on the order of the square of the product of the number of plans and the number of summary conditions per plan at that level of expansion. The solution space is the number of temporal orderings of the expanded plans (approximated by the factorial).

erations for that level. At level $d-2$, there are b^{d-2} plans, and the number of conditions that must be compared among their subplans at level $d-1$ additionally includes those propagated from the primitive level for a total of $3c' + b3c'$ conditions. Thus, $O(b^{d-2}b^2(c'+bc')^2)$ operations are performed at level $d-2$. This generalizes to $O(\sum_{i=0}^{d-1} b^i b^2 (b^{d-i-1}c')^2)$ operations for the entire hierarchy. We can reduce this to $O(b^2c'^2 \sum_{i=0}^{d-1} b^{2d-i-2}) = O(b^{2d}c'^2)$, and since $n = O(b^d)$, the complexity can be simply stated as $O(n^2c'^2)$.

In this worst case, the number of summary conditions for an abstract plan grows exponentially as you go up the hierarchy as shown in the second column of Figure 3. At the primitive level d, each plan has only $3c' = O(c')$ conditions, and there are $c' + bc' = O(bc')$ summary conditions for each plan at level $d-1$ and $O(b^2c')$. There are at most $3nc'$, or $O(b^d c')$, summary conditions at the root of the hierarchy—this is the total number of pre-, in-, and postconditions in the hierarchy. One might argue that in such cases deriving summary information only increases computation. But, actually, exponential computation time is saved when decisions based on summary information can be made at abstract levels because the complexity from exponential growth in the number of plans down the hierarchy outweighs the complexity of conditions growing exponentially up the hierarchy. This is because, as will be shown, the only known al-

gorithms for synchronizing plan steps to avoid conflicts are exponential with respect to the number of plans expanded in the hierarchy, which also grows exponentially with the depth. This exponential growth down the hierarchy outweighs the exponential growth of summary conditions in plans up the hierarchy. So the improvements made using summary information can yield exponential savings while only incurring a small polynomial overhead in deriving and using summary information.

Let's make this more clear. At the ith depth level in the hierarchy, each of the $O(b^i)$ plans has $O(b^{d-i}c'^2)$ summary conditions in the worst case. As described in [2], the algorithm to check whether a particular ordering of n plan steps (each with c summary conditions) results in all plans executing successfully is similar to deriving their collective summary information and has a complexity of $O(n^2c^2)$. Checking such a synchronization for the plans at any level i in a plan hierarchy is, thus, $O(b^{2i}b^{2d-2i}c'^2) = O(b^{2d}c'^2)$. So, since i drops out, the complexity of doing this check is *independent of the depth level*. In Figure 3, this is shown in the fifth column of the table where the number of operations is the same at each level. But, there is a huge space of $n! = O((b^i)!)$ sequential orderings[3] of the n plans at level i to potentially check to find a valid synchronization.[4] Thus, the search space grows doubly exponentially down the hierarchy despite the worst case when the number of conditions grows exponentially up the hierarchy. This argument assumes that finding a valid synchronization is intractable for larger numbers of plan steps, so we show that it is actually NP-complete. We reduce HAMILTONIAN PATH to the THREAT RESOLUTION problem for STRIPS planning and claim that a similar reduction can be done for our problem that allows concurrent execution.

Theorem THREAT RESOLUTION is NP-complete. This is the problem of determining whether there is a set of ordering constraints that can be added to a partial order STRIPS plan such that no operator's preconditions are threatened by another operator's effects.

Proof If there is a set of ordering constraints that will resolve all threats, then there is at least one corresponding total order where there are no threats. Thus, the problem is in NP since orderings of operators can be chosen non-deterministically, and threats can be identified in polynomial time.

For a directed graph $G = (V, E)$ with nodes $v_1, v_2, \ldots, v_n \in V$ and edges $e_1, e_2, \ldots, e_m \in E$ (a set of ordered pairs of nodes), HAMILTONIAN PATH is the problem that asks if there is a path that visits each node exactly once. We build an instance of THREAT RESOLUTION (a partial order plan) by creating an operator for each node v_i. The only precondition of the operator is $A(i)$, representing the *accessibility* of the node. There is a postcondition $A(j)$ for each edge $e_k = (v_i, v_j)$, and a postcondition $\overline{A(l)}$ for all other nodes for which there is no edge from v_i. All operators are unordered and the initial state and goal state is empty.

[3] There are more for other orderings allowing for concurrent execution.

[4] This is why Georgeff[7] chose to cluster multiple operators into "critical regions" and synchronize the (fewer) regions since there would be many fewer interleavings to check. By exploiting the hierarchical structure of plans, we use the "clusters" predefined in the hierarchy to this kind of advantage without needing to cluster from the bottom up.

If there is a Hamiltonian path for the graph, then the operators for the nodes can be ordered the same as the nodes in the path because the accessibility preconditions of each operator will be satisfied by the previous operator. If there is no Hamiltonian path for the graph, then there is no consistent ordering of the operators. We know this because there is a one-to-one mapping from an ordering of nodes to an ordering of operators. If the ordering of the nodes is such that there is no edge from one to a succeeding node, then the accessibility precondition of the corresponding operator will be clobbered. In addition, for any walk through the graph, there eventually will be an unvisited node for which there is no edge from the last node visited. In this case, the unvisited node will be clobbered because its accessibility precondition will not be met. Thus, THREAT RESOLUTION is NP-hard, and since it was shown to be in NP, it is NP-complete. □

In order to show that resolving threats among CHiPs is also NP-complete, we only need to add inconditions to each operator that prevent concurrent action. This can be done by adding $A(i)$ for v_i and $\overline{A(j)}$ for every other $v_j \in V$ to the inconditions of the operator corresponding to v_i for each $v_i \in V$. This ensures that the only temporal relations that can hold between any pair of operators are $before$, $after$, $meets$, or $imeets$, and the one-to-one mapping from paths in the graph to sequences of operators is preserved.

There are only and plans in this worst case. In the case that there are or plans, by similar argument, being able to prune branches at higher levels based on summary information will greatly improve the search despite the overhead of deriving and using summary conditions. Obviously, the computational savings of using summary information will be even greater when there are conditions common to plans on the same level, and the number of summary conditions does not grow exponentially up the hierarchy. Still, surely there are cases where none of the details of the plan hierarchy can be ignored, and summary information would incur unnecessary overhead, but when the size of problem instances are scaled, dealing with these details will likely be infeasible anyway.

4 Experiments

The experiments described here used the coordination algorithm described in Section 2.4 with all of the stated heuristics. It was compared to another top-down search algorithm that did not use summary information but used a FAF ("fewest alternatives first") heuristic [4] to decide the order in which *or* subplans are investigated. This simply means we chose to expand the *or* subplan that had the fewest number of subplan choices. Since no summary information was used, threats could only be resolved at primitive levels. The FAF heuristic has been shown to be effective in the HTN planning domain to get large improvements in search time [8], and a similar approach to ours shows how heuristics using external conditions can be used to get exponential improvements over FAF [9]. We show here that summary information can also be used to gain significant improvements over FAF. Certainly, a comparison of our approach with that in [9] could help shed light on the benefits and disadvantages of varying amounts of summary information. This is a future consideration of this work.

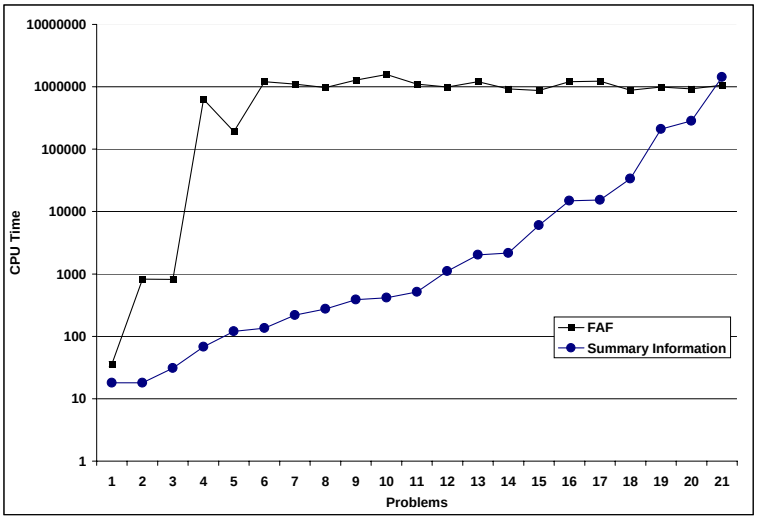


Fig. 4. CPU time measurements comparing summary information heuristics to FAF for finding optimal solutions. FAF only solved problems 1-5 and 7; others were killed when the search queue was too large to fit in memory.

The problems were hand-crafted from the NEO domain, described in the example in the Introduction. Agents had plans to either visit their specified locations by traveling in one direction only or switching directions at some location. These choices expand into choices to begin traveling clockwise or counterclockwise. For the branch where the agent switches directions, it can choose to change directions at any location it is specified to visit. Primitive actions are to move between adjacent locations without running into another agent. Optimality is measured as the total completions time where each move has a uniform time cost. We chose problems with four, six, and eight locations; with two and three agents; and with no, some, and complete overlap in the locations the agents visited. Results of the experiments are given in Figure 4.

For problems with only four locations and two agents, both algorithms found the optimal solution quickly. For more complex algorithms, the heuristics using summary information appear to make great improvements over FAF, which could only solve six of the 21 problems within memory constraints. These results are by no means conclusive, but they do show promise for search based on summary information. For most of these problems, coordinating at the primitive level was intractable. In most cases, the algorithm using summary information was able to find an abstract solution quickly.

5 Conclusions and Future Work

We have shown that summary information can can find solutions at higher levels exponentially more quickly than at lower levels; and we have identified heuristics and search techniques that can take advantage of summary information in finding coordinated plans. In addition, we have characterized a coordination algorithm that takes advantage of these search techniques and experimentally shown how it can make large improvements over an FAF heuristic in finding optimal coordinated plans. More work is needed to show that these results translate to different domains, and future considerations include comparing this approach to other planning heuristics that capitalize on domain knowledge in order to better understand the relationship between plan structure and search performance. We expect the benefits of using summary information to also apply to hierarchical planning and wish to compare these techniques with current heuristics for concurrent hierarchical planning.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
2. B. Clement and E. Durfee. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*, 1999.
3. B. Clement and E. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. Intl. Conf. Autonomous Agents*, 1999.
4. K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.
5. E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI*, pages 375–380, July 1994.
6. R. E. Fikes and Nilsson N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
7. M. P. Georgeff. Communication and interaction in multiagent planning. In *Proc. AAAI*, pages 125–129, 1983.
8. R. Tsuneto, J. Hendler, and D. Nau. Space-size minimizationin refinement planning. In *Proc. Fourth European Conference on Planning*, 1997.
9. R. Tsuneto, J. Hendler, and D. Nau. Analyzing external conditions to improve the efficiency of htn planning. In *Proc. AAAI*, pages 913–920, 1998.
10. D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

Appendix E: Using Abstraction in Planning and Scheduling

Bradley J. Clement[1], Anthony C. Barrett[1], Gregg R. Rabideau[1], and Edmund H. Durfee[2]

[1] Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 126-347, Pasadena, CA 91109-8099 USA
{bclement, barrett, rabideau}@aig.jpl.nasa.gov
[2] Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110 USA
durfee@umich.edu

Abstract. We present an algorithm for summarizing the metric resource requirements of an abstract task based on the resource usages of its potential refinements. We use this summary information within the ASPEN planner/scheduler to coordinate a team of rovers that conflict over shared resources. We find analytically and experimentally that an iterative repair planner can experience an exponential speedup when reasoning with summary information about resource usages and state constraints, but there are some cases where the extra overhead involved can degrade performance.

1 Introduction

Hierarchical Task Network (HTN) planners [4] represent abstract actions that decompose into choices of action sequences that may also be abstract, and HTN planning problems are requests to perform a set of abstract actions given an initial state. The planner subsequently refines the abstract tasks into less abstract subtasks to ultimately generate a schedule of primitive actions that is executable from the initial state. This differs from STRIPS planning where a planner can find any sequence of actions whose execution can achieve a set of goals. HTN planners only find sequences that perform abstract tasks and a domain expert can intuitively define hierarchies of abstract tasks to make the planner rapidly generate all sequences of interest.

Previous research [10, 9] has shown that, under certain restrictions, hierarchical refinement search reduces the search space by an exponential factor. Subsequent research has shown that these restrictions can be dropped by reasoning during refinement about the conditions embodied by abstract actions [3, 2]. These *summarized conditions* represent the internal and external requirements and effects of an abstract action and those of any possible primitive actions that it can decompose into. Using this information, a planner can detect and resolve conflicts between abstract actions and sometimes can find abstract solutions or determine that particular decomposition choices are inconsistent. In this paper, we apply these abstract reasoning techniques to tasks that use metric resources. We present an algorithm that processes a task hierarchy description offline to summarize abstract plan operators' metric resource requirements.

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. This work was also supported in part by DARAP(F30602-98-2-0142).

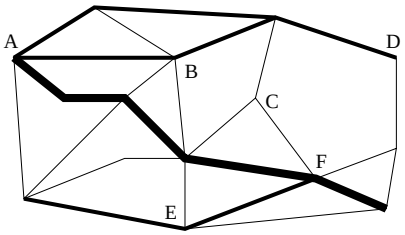


Fig. 1. Example map of established paths between points in a rover domain, where thinner edges are harder to traverse, and labeled points have associated observation goals

While planning and scheduling efficiency is a major focus of our research, another is the support of flexible plan execution systems such as PRS [6], UMPRS [11], RAPS [5], JAM [7], etc., that exploit hierarchical plan spaces while interleaving task decomposition with execution. By postponing task decomposition, such systems gain flexibility to choose decompositions that best match current circumstances. However, this means that refinement decisions are made and acted upon before all abstract actions are decomposed to the most detailed level. If such refinements at abstract levels introduce unresolvable conflicts at more detailed levels, the system will get stuck part way through executing the tasks to perform the requested abstract tasks. By using summary information, a system that interleaves planning and execution can detect and resolve conflicts at abstract levels to avoid getting stuck and to provide some ability to recover from failure.

In the next section this paper uses a traveling rover example to describe how we represent abstract actions and summary information. Given these representations, the subsequent section presents an algorithm for summarizing an abstract task's potential resource usage based on its possible refinements. Next we analytically show how summary information can accelerate an iterative repair planner/scheduler and make some empirical measurements in a multi-rover planning domain.

2 Representations

To illustrate our approach, we will focus on managing a collection of rovers as they explore the environment around a lander on Mars. This exploration takes the form of visiting different locations and making observations. Each traversal between locations follows established paths to minimize effort and risk. These paths combine to form a network like the one mapped out in Figure 1, where vertices denote distinguished locations, and edges denote allowed paths. While some paths are over hard ground, others are over loose sand where traversal is harder since a rover can slip.

2.1 Resources and Tasks

More formally, we represent each rover's status in terms of state and resource variables. The values in state variables record the status of key rover subsystems. For instance, a rover's *position* state variable can take on the label of any vertex in the location network. Given this representation of state information, tasks have preconditions/effects that we represent as equality constraints/assignments. In our rover example traveling on the arc from point A to point B is done with a *go(A,B)* task. This task has the precondition $(position{=}A)$ and the effect $(position{=}B)$.

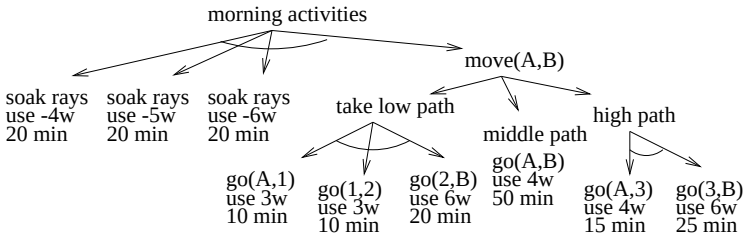


Fig. 2. AND/OR tree defining abstract tasks and how they decompose for a morning drive from point A to point B along one of the three shortest paths in our example map

In addition to interacting with state variables, tasks use resources. While some resources are only used during a task, using others persists after a task finishes. The first type of resource is *nondepletable*, with examples like solar power which immediately becomes available after some task stops using it. On the other hand, battery energy is a *depletable* resource because its consumption persists until a later task recharges the battery. We model a task's resource consumption by subtracting the usage amount from the resource variable when the task starts and for nondepletable resources adding it back upon completion. While this approach is simplistic, it can conservatively approximate any resource consumption profile by breaking a task into smaller subtasks.

Primitive tasks affect state and resource variables, and an abstract task is a non-leaf node in an AND/OR tree of tasks.[1] An AND task is executed by executing all of its subtasks according to a some set of specified temporal constraints. An OR task is executed by executing one of its subtasks. Figure 2 gives an example of such an abstract task. Imagine a rover that wants to make an early morning trip from point A to point B on our example map. During this trip the sun slowly rises above the horizon giving the rover the ability to progressively use *soak rays* tasks to provide more solar power to motors in the wheels. In addition to collecting photons, the morning traverse moves the rover, and the resultant *go* tasks require path dependent amounts of power. While a rover traveling from point A to point B can take any number of paths, the shortest three involve following one, two, or three steps.

2.2 Summary Information

An abstract task's state variable summary information includes elements for pre-, in-, and postconditions. Summary preconditions are conditions that must be met by the initial state or previous external tasks in order for a task to decompose and execute successfully, and a task's summary postconditions are the effects of its decomposition's execution that are not undone internally. We use summary inconditions for those conditions that are required or asserted in the task's decomposition during the interval of execution. All summary conditions are used to reason about how state variables are affected while performing an abstract task, and they have two orthogonal types of modalities:

- $must$ or may indicates that a condition holds in all or some decompositions of the abstract task respectively and
- $first$, $last$, $sometimes$, or $always$ indicates when a condition holds in the task's execution interval.

[1] It is trivial to extend the algorithms in this paper to handle state and resource constraints specified for abstract tasks.

For instance, the $move(A, B)$ task in our example has a $must, first(position{=}A)$ summary precondition and a $must, last(position{=}B)$ postcondition because all decompositions move the rover from A to B. Since the $move(A, B)$ task decomposes into one of several paths, it has summary inconditions of the form $may, sometimes(position{=}i)$, where i is 1, 2 or 3. State summary conditions are formalized in [2].

Extending summary information to include metric resources involves defining a new representation and algorithm for summarization. A ***summarized resource usage*** consists of ranges of potential resource usage amounts during and after performing an abstract task, and we represent this summary information using the structure

$$\langle local\_min\_range, local\_max\_range, persist\_range \rangle,$$

where the resource's local usage occurs within the task's execution, and the persistent usage represents the usage that lasts after the task terminates for depletable resources.

The usage ranges capture the multiple possible usage profiles of an task with multiple decomposition choices and timing choices among loosely constrained subtasks. For example, the *high path* task has a $\langle [4, 4], [6, 6], [0, 0] \rangle$ summary power use over a 40 minute interval. In this case the ranges are single points due to no uncertainty – the task simply uses 4 watts for 15 minutes followed by 6 watts for 25 minutes. The $move(A, B)$ provides a slightly more complex example due to its decompositional uncertainty. This task has a $\langle [0, 4], [4, 6], [0, 0] \rangle$ summary power use over a 50 minute interval. In both cases the $persist\_range$ is $[0, 0]$ because power is a nondepletable resource.

While a summary resource usage structure has only one range for persistent usage of a resource, it has ranges for both the minimum and maximum local usage because resources can have minimum as well as maximum usage limits, and we want to detect whether a conflict occurs from violating either of these limits. As an example of reasoning with resource usage summaries, suppose that only 3 watts of power were available during a $move(A, B)$ task. Given the $[4, 6]$ $local\_max\_range$, we know that there is an unresolvable problem without decomposing further. Raising the available power to 4 watts makes the task executable depending on how it gets decomposed and scheduled, and raising to 6 or more watts makes the task executable for all possible decompositions.

3 Resource Summarization Algorithm

The state summarization algorithm [2] recursively propagates summary conditions upwards from an AND/OR tree's leaves, and the algorithm for resource summarization takes the same approach. Starting at the leaves, we find primitive tasks that use constant amounts of a resource. The resource summary of a task using x units of a resource is $\langle [x,x],[x,x],[0,0] \rangle$ or $\langle [x,x],[x,x],[x,x] \rangle$ over the task's duration for nondepletable or depletable resources respectively.

Moving up the AND/OR tree we either come to an AND or an OR branch. For an OR branch the combined summary usage comes from the OR computation

$$\begin{array}{l} \langle [min_{c \in children}(lb(local\_min\_range(c))), \\ \quad max_{c \in children}(ub(local\_min\_range(c)))], \\ \quad [min_{c \in children}(lb(local\_max\_range(c))), \\ \quad max_{c \in children}(ub(local\_max\_range(c)))], \\ \quad [min_{c \in children}(lb(persist\_range(c))), \\ \quad max_{c \in children}(ub(persist\_range(c)))] \rangle, \end{array}$$

where $lb()$ and $ub()$ extract the lower bound and upper bound of a range respectively. The $children$ denote the branch's children with their durations extended to the length of the

longest child. This duration extension alters a child's resource summary information because the child's usage profile has a 0 resource usage during the extension. For instance, when we determine the resource usage for $move(A, B)$ we combine two 40 minute tasks with a 50 minute task. The resulting summary information is for a 50 minute abstract task whose profile might have a zero watt power usage for 10 minutes. This extension is why $move(A, B)$ has a $[0, 4]$ for a $local\_min\_range$ instead of $[3, 4]$. Planners that reason about variable durations could use $[3, 4]$ for a duration ranging from 40 to 50.

Computing an AND branch's summary information is a bit more complicated due to timing choices among loosely constrained subtasks. Our *take x path* examples illustrate the simplest subcase, where subtasks are tightly constrained to execute serially. Here profiles are appended together, and the resulting summary usage information comes form the SERIAL-AND computation

$$\begin{array}{l}\langle[min_{c\in children}(lb(local\_min\_range(c)) + \Sigma^{pre}_{lb}(c)),\\ \ min_{c\in children}(ub(local\_min\_range(c)) + \Sigma^{pre}_{ub}(c))],\\ \ [max_{c\in children}(lb(local\_max\_range(c)) + \Sigma^{pre}_{lb}(c)),\\ \ max_{c\in children}(ub(local\_max\_range(c)) + \Sigma^{pre}_{ub}(c))],\\ \ [\Sigma_{c\in children}(lb(persist\_range(c))),\\ \ \Sigma_{c\in children}(ub(persist\_range(c)))]\rangle,\end{array}$$

where $\Sigma^{pre}_{lb}(c)$ and $\Sigma^{pre}_{ub}(c)$ are the respective lower and upper bounds on the cumulative persistent usages of children that execute before c. These computations have the same form as the Σ computations for the final $persist\_range$.

The case where all subtasks execute in parallel and have identical durations is slightly simpler. Here the usage profiles add together, and the branch's resultant summary usage comes from the PARALLEL-AND computation

$$\begin{array}{l}\langle[\Sigma_{c\in children}(lb(local\_min\_range(c))),\\ \ max_{c\in children}(ub(local\_min\_range(c)) + \Sigma^{non}_{ub}(c))],\\ \ [min_{c\in children}(lb(local\_max\_range(c)) + \Sigma^{non}_{lb}(c)),\\ \ \Sigma_{c\in children}(ub(local\_max\_range(c)))],\\ \ [\Sigma_{c\in children}(lb(persist\_range(c))),\\ \ \Sigma_{c\in children}(ub(persist\_range(c)))]\rangle,\end{array}$$

where $\Sigma^{non}_{ub}(c)$ and $\Sigma^{non}_{lb}(c)$ are the respective sums of $local\_max\_range$ upper bounds and $local\_min\_range$ lower bounds for all children except c.

To handle AND tasks with loose temporal constraints, we consider all legal orderings of child task endpoints. For example, in our rover's early morning tasks, there are three serial solar energy collection subtasks running in parallel with a subtask to drive to location B. Figure 3 shows one possible ordering of the subtask endpoints, which breaks the $move(A, B)$ into three pieces, and two of the *soak rays* children in half. Given an ordering, we can (1) use the endpoints of the children to determine subintervals, (2) compute summary information for each child task/subinterval combination, (3) combine the parallel subinterval summaries using the PARALLEL-AND computation, and then (4) chain the subintervals together using the SERIAL-AND computation. Finally, the AND task's summary is computed by combining the summaries for all possible orderings using an OR computation.

Here we describe how step (2) generates different summary resource usages for the subintervals of a child task. A child task with summary resource usage $\langle[a,b],[c,d],[e,f]\rangle$ contributes one of two summary resource usages to each intersecting subinterval [2]:

$$\langle[a, b], [c, d], [0, 0]\rangle, \langle[a, d], [a, d], [0, 0]\rangle.$$

[2] For summary resource usages of the last interval intersecting the child task, we replace $[0, 0]$ with $[e, f]$ in the $persist\_range$.

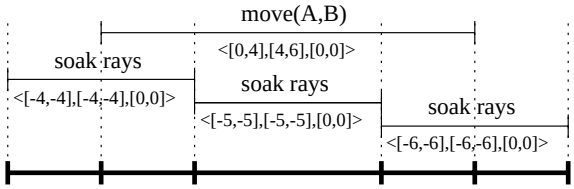


Fig. 3. Possible task ordering for a rover's morning activities, with resulting subintervals.

While the first usage has the tighter $[a, b], [c, d]$ local ranges, the second has looser $[a, d], [a, d]$ local ranges. Since the b and c bounds only apply to the subintervals containing the subtask's minimum and maximum usages, the tighter ranges apply to one of a subtask's intersecting subintervals. While the minimum and maximum usages may not occur in the same subinterval, symmetry arguments let us connect them in our computation. Thus one subinterval has tighter local ranges and all other intersecting subintervals get the looser local ranges, and the extra complexity comes from having to investigate all subtask/subinterval assignment options. For instance, there are three subintervals intersecting $move(A, B)$ in Figure 3, and three different assignments of summary resource usages to the subintervals: placing $[0, 4], [4, 6]$ in one subinterval with $[0, 6], [0, 6]$ in the other two. These placement options result in a subtask with n subintervals having n possible subinterval assignments. So if there are m child tasks each with n alternate assignments, then there are n^m combinations of potential subtask/subinterval summary resource usage assignments. Thus propagating summary information through an AND branch is exponential in the number of subtasks with multiple internal subintervals. However since the number of subtasks is controlled by the domain modeler and is usually bounded by a constant, this computation is tractable. In addition, summary information can often be derived offline for a domain. The propagation algorithm takes on the form:

- For each consistent ordering of endpoints:
  - For each consistent subtask/subinterval summary usage assignment:
    - Use PARALLEL-AND computations to combine subtask/subinterval summary usages by subinterval.
    - Use a SERIAL-AND computation on the subintervals' combined summary usages to get a consistent summary usage.
- Use OR computation to combine all consistent summary usages to get AND task's summary usage.

4 Using Summary Information

In this section, we describe techniques for using summary information in local search planners to reason at abstract levels effectively and discuss the complexity advantages. Reasoning about abstract plan operators using summary information can result in exponential planning performance gains for backtracking hierarchical planners [3]. In iterative repair planning, a technique called *aggregation* that involves scheduling hierarchies of tasks similarly outperforms the movement of tasks individually [8]. But, can summary information be used in an iterative repair planner to improve performance when aggregation is already used? We demonstrate that summarized state and resource constraints makes exponential improvements by collapsing constraints at abstract levels. First, we describe how we use aggregation and summary information to schedule tasks within an

iterative repair planner. Next, we analyze the complexity of moving abstract and detailed tasks using aggregation and summary information. Then we describe how a heuristic iterative repair planner can exploit summary information.

4.1 Aggregation and Summary Information

While HTN planners commonly take a generative least commitment approach to problem solving, research in the OR community illustrates that a simple local search is surprisingly effective [12]. Heuristic iterative repair planning uses a local search to generate a plan. It starts with an initial flawed plan and iteratively chooses a flaw, chooses a repair method, and changes the plan by applying the method. Unlike generative planning, the local search never backtracks. Since taking a random walk through a large space of plans is inefficient, heuristics guide the choices by determining the probability distributions for each choice. We build on this approach to planning by using the ASPEN planner [1].

Moving tasks is a central scheduling operation in iterative repair planners. A planner can more effectively schedule tasks by moving related groups of tasks to preserve constraints among them. Hierarchical task representations are a common way of representing these groups and their constraints. Aggregation involves moving a fully detailed abstract task hierarchy while preserving the temporal ordering constraints among the subtasks. Moving individual tasks independent of their siblings and subtasks is shown to be much less efficient [8]. Valid placements of the task hierarchy in the schedule are computed from the state and resource usage profile for the hierarchy. This profile represents one instantiation of the decomposition and temporal ordering of the abstract task's hierarchy.

A summarized state or resource usage represents all potential profiles of an abstract task before it is decomposed. Our approach involves reasoning about summarized constraints in order to schedule abstract tasks before they are decomposed. Scheduling an abstract task is computationally cheaper than scheduling the task's hierarchy using aggregation when the summarized constraints more compactly represent the constraint profiles of the hierarchy. This improves the overall performance when the planner/scheduler resolves conflicts and finds solutions at abstract levels before fully decomposing tasks.

4.2 Complexity Analysis

To move a hierarchy of tasks using aggregation, valid intervals must be computed for each resource variable affected by the hierarchy.[3] These valid intervals are intersected for the valid placements for the abstract tasks and their children. The complexity of computing the set of valid intervals for a resource is $O(cC)$ where c is the number of constraints (usages) an abstract task has with its children for the variable, and C is the number of constraints of other tasks in the schedule on the variable [8]. If there are n similar task hierarchies in the entire schedule, then $C = (n-1)c$, and the complexity of computing valid intervals is $O(nc^2)$. But this computation is done for each of v resource variables (often constant for a domain), so moving a task will have a complexity of $O(vnc^2)$.

The summary information of an abstract task represents all of the constraints of its children, but if the children share constraints over the same resource, this information is collapsed into a single *summary* resource usage in the abstract task. Therefore, when moving an abstract task, the number of different constraints involved may be far fewer depending on the domain. If the scheduler is trying to place a summarized abstract

[3] The analysis also applies to state constraints, but we restrict our discussion to resource usage constraints for simplicity.

task among other summarized tasks, the computation of valid placement intervals can be greatly reduced because the c in $O(vnc^2)$ is smaller. We now consider two extreme cases where constraints can be fully collapsed and where they cannot be collapsed at all.

In the case that all tasks in a hierarchy have constraints on the same resource, the number of constraints in a hierarchy is $O(b^d)$ for a hierarchy of depth d and branching factor (number of child tasks per parent) b. In aggregation, where hierarchies are fully detailed first, this means that the complexity of moving an task is $O(vnb^{2d})$ because $c = O(b^d)$. Now consider using aggregation for moving a partially expanded hierarchy where the leaves are summarized abstract tasks. If all hierarchies in the schedule are decomposed to level i, there are $O(b^i)$ tasks in a hierarchy, each with one summarized constraint representing those of all of the yet undetailed subtasks beneath it for each constraint variable. So $c = O(b^i)$, and the complexity of moving the task is $O(vnb^{2i})$. Thus, moving an abstract task using summary information can be a factor of $O(b^{2(d-i)})$ times faster than for aggregation.

The other extreme is when all of the tasks place constraints on different variables. In this case, $c = 1$ because any hierarchy can only have one constraint per variable. Fully detailed hierarchies contain $v = O(b^d)$ different variables, so the complexity of moving a task in this case is $O(nb^d)$. If moving a summarized abstract task where all tasks in the schedule are decomposed to level i, v is the same because the abstract task summarizes all constraints for each subtask in the hierarchy beneath it, and each of those constraints are on different variables such that no constraints combine when summarized. Thus, the complexity for moving a partially expanded hierarchy is the same as for a fully expanded one. Experiments in Section 5 exhibit great improvement for cases when tasks have constraints over common resource variables.

Along another dimension, scheduling summarized tasks is exponentially faster because there are fewer *temporal* constraints among higher level tasks. When task hierarchies are moved using aggregation, all of the local temporal constraints are preserved. However, there are not always valid intervals to move the entire hierarchy. Even so, the scheduler may be able to move less constraining lower level tasks to resolve the conflict. In this case, temporal constraints may be violated among the moved task's parent and siblings. The scheduler can then move and/or adjust the durations of the parent and siblings to resolve the conflicts, but these movements can affect higher level temporal constraints or even produce other conflicts. At a depth level i in a hierarchy with decompositions branching with a factor b, the task movement can affect b^i siblings in the worst case and produce a number of conflicts exponential to the depth of the task. Thus, if all conflicts can be resolved at an abstract level i, $O(b^{d-i})$ scheduling operations may be avoided. In Section 5, empirical data shows the exponential growth of computation with respect to the depth at which ASPEN finds solutions.

Other complexity analyses have shown that under certain restrictions different forms of hierarchical problem solving can reduce the size of the search space by an exponential factor [10, 9]. Basically, these restrictions are that an algorithm never needs to backtrack from lower levels to higher levels in the problem. In other words, subproblems introduced in different branches of the hierarchy do not interact. We do not make this assumption for our problems. However, the speedup described above does assume that the hierarchies need not be fully expanded to find solutions.

4.3 Decomposition Heuristics for Iterative Repair

Despite this optimistic complexity, reasoning about summarized constraints only translates to better performance if the movement of summarized tasks resolves conflicts and

advances the search toward a solution. There may be no way to resolve conflicts among abstract tasks without decomposing them into more detailed ones. So when should summary information be used to reason about abstract tasks, and when and how should they be decomposed? Here, we describe techniques for reasoning about summary information as abstract tasks are detailed.

We explored two approaches that reason about tasks from the top-level of abstraction down in the manner described in [3]. Initially, the planner only reasons about the summary information of fully abstracted tasks. As the planner manipulates the schedule, tasks are gradually decomposed to open up new opportunities for resolving conflicts using the more detailed child tasks. One strategy (that we will refer to as *level-decomposition*) is to interleave repair with decomposition as separate steps. Step 1) The planner repairs the current schedule until the number of conflicts cannot be reduced. Step 2) It decomposes all abstract tasks one level down and returns to Step 1. By only spending enough time at a particular level of expansion that appears effective, the planner attempts to find the highest decomposition level where solutions exist without wasting time at any level.

Another approach is to use decomposition as one of the repair methods that can be applied to a conflict so that the planner gradually decomposes conflicting tasks. This strategy tends to decompose the tasks involved in more conflicts since any task involved in a conflict is potentially expanded when the conflict is repaired. The idea is that the scheduler can break overconstrained tasks into smaller pieces to offer more flexibility in rooting out the conflicts. This resembles the EMTF (expand-most-threats-first) [3] heuristic that expands (decomposes) tasks involved in more conflicts before others. (Thus, we later will refer to this heuristic as EMTF.) This heuristic avoids unnecessary reasoning about the details of non-conflicting tasks. This is similar to a most-constrained variable heuristic often employed in constraint satisfaction problems.

Another heuristic for improving planning performance prefers decomposition choices that lead to fewer conflicts. In effect, this is a least-constraining value heuristic used in constraint satisfaction approaches. Using summary information, the planner can test each child task by decomposing to the child and replacing the parent's summarized constraints that summarize the children with the particular child's summarized constraints. For each child, the number of conflicts in the schedule are counted, and the child creating the fewest conflicts is chosen.[4] This is the *fewest-threats-first* (FTF) heuristic that is shown to be effective in pruning the search space in a backtracking planner [3]. Likewise, the experiments in Section 5 show similar performance improvements.

5 Empirical Comparisons

The experiments we describe here show that summary information improves performance significantly when tasks within the same hierarchy have constraints over the same resource, and solutions are found at some level of abstraction. At the same time, we find cases where abstract reasoning incurs significant overhead when solutions are only found at deeper levels. However, in domains where decomposition choices are critical, we show that this overhead is insignificant because the FTF heuristic finds solutions at deeper levels with better performance. These experiments also show that the EMTF heuristic outperforms level-decomposition for certain decomposition rates. In addition, we show that the time to find a solution increases dramatically with the depth where solutions are found, supporting the analysis at the end of Section 4.2.

[4] Or, in stochastic planners like ASPEN, the children are chosen with probability decreasing with their respective number of conflicts.

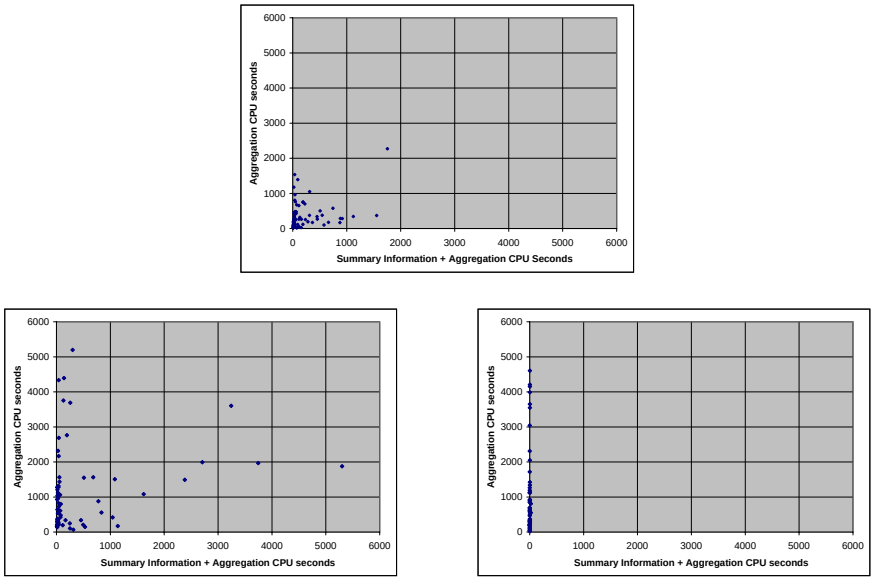


Fig. 4. Plots for the *no channel*, *mixed*, and *channel only* domains

The domain for our problems expands the single rover problem described in earlier sections to a team of rovers that must resolve conflicts over shared resources. Paths between waypoints are assigned random capacities such that either one, two, or three rovers can traverse a path simultaneously; only one rover can be at any waypoint; and rovers may not traverse paths in opposite directions. In addition, rovers must communicate with the lander for telemetry using a shared channel of fixed bandwidth. Depending on the terrain, the required bandwidth varies. 80 problems were generated for two to five rovers, three to six observation locations per rover, and 9 to 105 waypoints. Schedules ranged from 180 to 1300 tasks. Note that we use a prototype interface for summary information, and some of ASPEN's optimized scheduling techniques could not be used.

We compare ASPEN using aggregation with and without summarization for three variations of the domain. The use of summary information includes the EMTF and FTF decomposition heuristics. One domain excludes the communications channel resource (*no channel*); one excludes the path capacity restrictions (*channel only*); and the other includes all mentioned resources *mixed*). Since all of the movement tasks reserve the channel resource, we expect greater improvement in performance when using summary information according to the complexity analyses in the previous section. Tasks within a rover's hierarchy rarely place constraints on other variables more than once, so the *no channel* domain corresponds to the case where summarization collapses no constraints.

Figure 4 (top) exhibits two distributions of problems for the *no channel* domain. In most of the cases (points along the y-axis), ASPEN with summary information finds a solution quickly at some level of abstraction. However, in many cases, summary information performs notably worse (points along the x-axis). We find that for these problems finding a solution requires the planner to dig deep into the rovers' hierarchies, and once it decomposes the hierarchies to these levels, the difference in the additional time to find a solution between the two approaches is negligible Thus, the time spent reasoning about summary information at higher levels incurred unnecessary overhead. Previous work shows that this overhead is rarely significant in backtracking planners because sum-

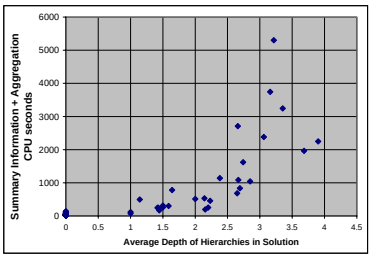


Fig. 5. CPU time for solutions found at varying depths.

mary information can prune inconsistent search spaces at abstract levels [3]. However, in non-backtracking planners like ASPEN, the only opportunity we found to prune the search space at abstract levels was using the FTF heuristic to avoid greater numbers of conflicts in particular branches. Later, we will explain why FTF is not helpful for this domain but very effective in a modified domain.

Figure 4 (left) shows significant improvement for summary information in the *mixed* domain compared to the *no channel* domain. Adding the channel resource rarely affected the use of summary information because the collapse in summary constraints incurred insignificant additional complexity. However, the channel resource made the scheduling task noticeably more difficult for ASPEN when not using summary information. In the *channel only* domain (Figure 4 right), summary information finds solutions at the abstract level almost immediately, but the problems are still complicated when ASPEN does not use summary information. These results support the complexity analysis in the previous section that argues that summary information exponentially improves performance when tasks within the same hierarchy make constraints over the same resource and solutions are found at some level of abstraction.

Figure 5 shows the CPU time required for ASPEN using summary information for the *mixed* domain for the depths at which the solutions are found. The depths are average depths of leaf tasks in partially expanded hierarchies. The CPU time increases dramatically for solutions found at greater depths, supporting our claim that finding a solution at more abstract levels is exponentially easier.

For the described domain, choosing different paths to an observation location usually does not make a significant difference in the number of conflicts encountered because if the rovers cross paths, all path choices will still lead to conflict. We created a new set of problems where obstacles force the rovers to take paths through corridors that have no connection to others paths. For these problems, path choices always lead down a different corridor to get to the target location, so there is usually a path that avoids a conflict and a path that causes one. The planner using the FTF heuristic dominates the planner choosing decompositions randomly for all but two problems (Figure 6 left).

Figure 6 (right) shows the performance of EMTF vs. level decomposition for different rates of decomposition for three problems selected from the set. The plotted points are averages over ten runs for each problem. Depending on the choice of rate of decomposition (the probability that a task will decompose when a conflict is encountered), performance varies significantly. However, the best decomposition rate can vary from problem to problem making it potentially difficult for the domain expert to choose. Our future work will include investigating the relation of decomposition rates to performance based on problem structure.[5]

[5] For other experiments, we used a decomposition rate of 20%.

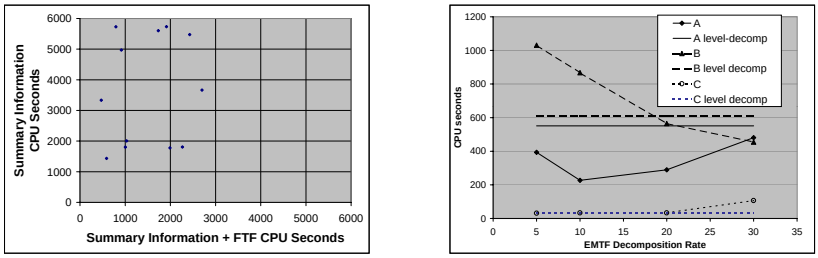


Fig. 6. Performance using FTF and EMTF vs. level-decomposition heuristics.

6 Conclusions

Reasoning about abstract constraints exponentially accelerates finding schedules when constraints collapse during summarization, and solutions at some level of abstraction can be found. Similar speedups occur when decomposition branches result in varied numbers of conflicts. The offline algorithm for summarizing metric resource usage makes these performance gains available for a larger set of expressive planners and schedulers. We have shown how these performance advantages can improve ASPEN's effectiveness when scheduling the tasks of multiple spacecraft. The use of summary information also enables a planner to preserve decomposition choices that robust execution systems can use to handle some degree of uncertainty and failure. Our future work includes evaluating the tradeoffs of optimizing plan quality using this approach as well as developing protocols to allow multiple spacecraft planners to coordinate their tasks asynchronously during execution.

References

1. S. Chien, G. Rabideu, R. Knight, R. Sherwood, B Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*, 2000.
2. B. Clement and E. Durfee. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*, pages 495–502, 1999.
3. B. Clement and E. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, pages 202–216, 2000.
4. K. Erol, J. Hendler, and D. Nau. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland, 1994.
5. J. Firby. *Adaptive Execution in Complex Dynamic Domains*. PhD thesis, Yale Univ., 1989.
6. M. Georgeff and A. Lansky. Procedural knowledge. *Proc. IEEE*, 74(10):1383–1398, Oct. 1986.
7. M. Huber. Jam: a bdi-theoretic mobile agent architecture. In *Proc. Intl. Conf. Autonomous Agents*, pages 236–243, 1999.
8. R. Knight, G. Rabideau, and S. Chien. Computing valid intervals for collections of activities with shared states and resources. In *Proc. AIPS*, pages 600–610, 2000.
9. C. Knoblock. Search reduction in hierarchical problem solving. In *Proc. AAAI*, pages 686–691, 1991.
10. R. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.
11. J. Lee, M. J. Huber, E. H. Durfee, and P. G. Kenny. Umprs: An implementation of the procedural reasoning system for multirobot applications. In *Proc. AIAA/NASA Conf. on Intelligent Robotics in Field, Factory, Service, and Space*, pages 842–849, March 1994.
12. Papadimitriou and Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications New York, 1998.

A Satisficing Multiagent Plan Coordination Algorithm for Dynamic Domains

[Extended Abstract]

Pradeep M. Pappachan
pmp@acm.org

Edmund H. Durfee
durfee@umich.edu

University of Michigan Artificial Intelligence Laboratory
1101 Beal Avenue, Ann Arbor, MI 48109-2110

1. INTRODUCTION

Coordinating agents need to anticipate how their individual actions will affect each other. Many proposed algorithms such as [4] have taken to searching through the joint plan space to find compatible action sequences *before* execution is begun. In dynamic environments, "reactive" strategies have been employed, where agents can wait until they must act to decide on their actions. However, waiting to decide on actions confounds coordination, since an action locally selected by an agent now might be incompatible with actions that others are taking or with actions that it or other agents might later discover they need to take. Proposed solutions include incremental merging of short term plans [3], institution of social laws [5], and exchanging post facto updates about agents' decisions on the fly and letting coordination trail behind [2]. Since these approaches have either focused on generating a *single* solution or pruning certain plan combinations from the solution space, they are not particularly suited to agents which might have to select from several alternative plans to achieve their tasks, since the approaches rule out options in the solution space prior to execution. In this paper, we propose an algorithm that coordinates agents with hierarchical task networks (HTNs) by using a least commitment strategy to incrementally construct a satisficing multiagent plan that respects task deadlines.

2. ASSUMPTIONS

In our approach, the coordination process is distributed between the task agents (each of which has one or more tasks to accomplish) and a coordinator agent. Each task agent has a hierarchical task network (HTN) which it executes in a top-down fashion by refining non-primitive operators to executable primitive operators through a sequence of operator reductions. Each agent has its own preferences with respect to the reduction of non-primitive operators at runtime. The coordinator, while not aware of these preferences, has the ability to compute the coordination implications of various sets of agent choices. We assume that the legal temporal relations between conflicting primitive operators from different HTNs are specified in terms of relations in Allen's interval algebra [1]. We also require that operator schemas in the HTNs be instantiated prior to execution.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

3. TASK CONFLICTS

Conflicts among plan operators arise in multiagent settings because an operator might undo the intended effects of another, or trigger conditions that make another impossible to execute. Conflicts can also arise when certain operators are executed concurrently. We say that operators U and V can potentially conflict under a temporal relation R, if and only if, the pre-, in-, or post-conditions associated with the two operators which are required to hold during overlapping time intervals (under the relational constraint R) are logically inconsistent. The objective of the coordinator is to temporally order agents' actions so as to preclude any potential conflicts between them.

4. TEMPORAL CONSTRAINT NETWORKS

The coordinator needs to keep track of the temporal ordering constraints between different plan operators as various coordination commitments are made at runtime. The primary data structure used for this purpose is the temporal constraint network. The coordinator is initially provided with a list of primitive plan operators that can potentially conflict, and for each pair in the list, a vector of atomic interval constraint relations under which the operators *do not* conflict. The temporal constraint network corresponding to a set of primitive operators includes vertices for every operator that can potentially conflict with at least one other operator in the set. The edges between the vertices are labeled with the vectors of atomic relations provided. Operators that are conflict-free have no ordering constraints relative to other operators, and are hence not represented in the network. It should be noted that two operators with incompatible contexts (necessary conditions for execution) may appear in the network. However, the edges between all such pairs of operators are omitted from the network.

5. CONSTRAINT RELAXATION

Consider two non-primitive operators U and V with the ordering constraint $\langle U, V, R_{UV} \rangle$, where R_{UV} is the relational ordering constraint between the operators. If operator U precedes V (or vice versa), it is clear that the operators in the reduction of U will precede those in the reduction of V (or vice versa). However, for any other instance of the temporal relation R_{UV}, there might be several ways of interleaving the operators in the reductions of U and V, corresponding to different sets of ordering constraints, each of which is temporally consistent with $\langle U, V, R_{UV} \rangle$ and the other temporal constraints represented in the network. The process of relaxing a constraint $\langle U, V, R \rangle$ consists of selecting a set of "new" ordering constraints $\langle u_i, V, R_{u_i V} \rangle$, (where u_i are operators in the reduction of U) that is consistent with $\langle U, V, R_{UV} \rangle$, and simultaneously satisfies certain completeness conditions. When such a set is selected, there is no "pressure" to satisfy the constraint $\langle U, V, R_{UV} \rangle$, and the constraint $\langle U, V, R_{UV} \rangle$ is said to be "relaxed". Each constraint relaxation step tightens the constraints between the operators in the temporal constraint network, and forces agents to temporally order their operators as they reduce them in a top-down fashion.

6. COORDINATION ALGORITHM

The Online Iterative Constraint Relaxation (OICR) algorithm takes the temporal constraint network for the plan coordination problem, and the top-level plan operators of the task agents as input. At the outset, the task agents block *only* their top-level operators from reduction (or execution). During the coordination process, several operators may in turn be blocked by the coordinator as it resolves potential conflicts between them. The agents are free to reduce or execute a blocked operator only when they receive an explicit "unblock" message from the coordinator for that operator. Initially, the coordinator sets feasible ordering constraints on the top-level operators, or aborts if no such constraints can be found. The top-level constraints are returned in a list, L' (say), which stores newly generated constraints during each iteration of the algorithm. Another list L keeps track of constraints that have not been relaxed.

If the list L' is empty, it signifies that there are no constraints between the top-level operators, and we are done; otherwise, at least one pair of top-level operators has an ordering constraint which needs to be relaxed. When a constraint needs to be relaxed, the coordinator requests the reduction of one of the operators in the constraint. The appropriate agent returns the reduction when it has decided which reduction to apply.

During each iteration, when the coordinator receives the reduction of some operator, U (say), that it had requested earlier in order to relax a constraint, $\langle U, V, R \rangle$ (say), in L, it applies the relaxation procedure to generate new constraints (between V and the operators in the reduction of U), which are then added to L'. (In some cases the constraint can be easily enforced by associating synchronization primitives with the operators, as in the case of the temporal relation *Precedes* and its inverse.) After the constraint has been relaxed, it can be deleted from the list L. The constraints in L' are then added to L, and they will be relaxed when, in future iterations, the requested reductions corresponding to those constraints arrive at the coordinator. Any operator in L' that is not in L is an operator that is not involved in any unrelaxed constraint; such operators are unblocked, and the task agents are free to reduce/execute them. The coordinator terminates when no unrelaxed constraints remain in L. At this point, the agents are fully coordinated, and if they execute their plans (as parts of the complete multiagent plan orchestrated by the coordinator), they will be able to achieve their goals.

In many real-world domains, it is necessary to look for coordinated plans which satisfy individual task deadlines. To generate satisficing multiagent plans that obey this requirement, the constraint relaxation procedure must only accept those sets of constraints that satisfy task deadlines. Given an HTN, the durations of primitive operators in the HTN, and a deadline for the top-level plan, we can regress the deadline through the primitive operators in the HTN to obtain deadlines for each primitive operator. We can also easily compute the earliest completion times for the primitives in the network. Given a pair of primitive operators (with deadlines), we can then compute for any atomic interval constraint, the change in the completion times of the operators induced by the constraint. This change in completion times is computed for all pairs of primitives in the network, and propagated through the network until the completion times of the primitives attain quiescence. If during this process, any primitive operator deadline is missed, we reject the set of constraints; otherwise, we can accept the constraints with the guarantee that they will not violate the deadlines.

7. EVALUATION

The OICR algorithm interleaves coordination with task execution, since operators that are not blocked can be reduced (or executed) while the coordinator is relaxing constraints. The coordinator does not have to examine entire HTNs before coordinating all the agents; it might terminate coordination at any level in the hierarchy. The coordination algorithm is provably sound, but not complete. The OICR algorithm has been implemented, and initial experiments in a peace-keeping coalition domain have shown that it can be used to efficiently coordinate small groups of agents with complex hierarchical task networks.

8. REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions of Systems, Man and Cybernetics*, 21(5):1167–1183, 1991.

[3] E. Ephrati and J. S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of AAAI*, pages 375–380, 1994.

[4] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proceedings of AAAI*, pages 125–129, 1983.

[5] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial societies. In *Proceedings of AAAI*, pages 276–281, 1992.

Appendix G:

Discovering and Exploiting Synergy Between Hierarchical Planning Agents

Jeffrey S. Cox
jeffcox@umich.edu

Edmund H. Durfee
durfee@umich.edu

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109

ABSTRACT

It is critical for agents in a multiagent environment to avoid interfering with each other when carrying out their tasks. However, to avoid execution inefficiencies, they also should capitalize on cooperative opportunities. In state oriented domains [14], identifying overlapping effects between agents' plans enables some agents to leave some tasks to others, thereby reducing the cost of execution and improving the overall efficiency of the multiagent system. This is what we term *synergy*. In this paper, we define criteria for finding a certain type of synergy involving agents with overlapping goals. We also develop algorithms for discovering this synergy between planning agents that exploit hierarchical plan representations. Our results show that our approach not only can reduce the costs of finding synergies compared to non-hierarchical strategies, but can also find synergies that might otherwise be missed.

Categories and Subject Descriptors

I.2.3 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Coordination of multiple agents, Multiagent planning, Plan merging, Synergy

1. INTRODUCTION

When cooperative agents must operate in a common multiagent environment, often the first task is to ensure that each agent will not cause delay or harm to other agents. Indeed, considerable research and effort has gone into developing multiagent planning systems to allow agents to anticipate and avoid unintended negative interactions, or conflicts [2, 9, 10, 13].

In addition to resolving conflicts, many researchers [12, 17, 18], have also explored how agents can identify and exploit positive interactions, or *synergies*, between their plans. Opportunities for multiagent plan synergy enable agents to reduce the effort they exert to achieve their goals. In particular, synergies exist when agents achieve overlapping or *subsuming* effects when trying to achieve their goals. In this case, the agents with the subsuming and subsumed plan steps can "merge" their plans (effectively allowing one to drop some of its plan steps in its plan) to reduce their combined cost of execution in state oriented domains [14].

An example of interacting agents would be two helicopter agents, operating on a different chain of commands, tasked to scout overlapping regions. There may be significant risks of collision, but these can be overcome by adequate conflict resolution planning. On the other hand, identifying overlapping plans (e.g., scouting the same locations) and merging certain duplicated tasks would allow each helicopter to accomplish its goals more efficiently.

Unfortunately, searching for synergies between agents can be as computationally costly as searching for conflicts. Furthermore, unlike conflict resolution, exploiting synergies is not a requirement for correct agent execution. Thus, while the payoff for discovering and resolving conflicts (e.g., helicopter crashes) may easily justify the often high costs of coordination, the benefits of finding and exploiting synergies are less often worth the costs. For example, the scouting helicopters might perform tasks for each other, cutting down on the overall flight times and fuel usages, but their missions might be delayed while spending time searching for these more cost-effective plans. Finding ways to reduce the cost of discovering synergies is thus of critical importance if such synergies are to be useful in practice.

Toward this end, we describe in this paper a novel approach to synergy search, the significance of which is the exploitation of hierarchical structures that makes the search tractable without having to predefine the opportunities for synergy. We show experimentally that our approach can find simple synergies between agents' plans more quickly than an approach that does not exploit plan hierarchies. We also show that the use of plan hierarchies allows our method to find these synergies between more abstract plan steps that would be overlooked with a "flat" (non-hierarchical)

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

approach, a crucial capability in applications where agents might share higher-level goals but not detailed capabilities.

We organize our paper as follows: In the next section, we summarize some of the concepts in plan merging, hierarchical plan representation, and multiagent planning that we use as a foundation for our approach. Section 3 details our new Synergy Algorithm, which is capable of identifying and exploiting plan synergy relationships between different hierarchical planning agents. In Section 4, we present some of our initial experimental results showing the advantages of our algorithm in terms of the main metrics we have laid out: that it is faster than a non-hierarchical approach at finding synergies, and that it can find synergies that a non-hierarchical approach will miss. Section 5 presents our conclusions and future directions of research.

2. RELATED WORK

Single agent plan merging has a rich history. Many researchers have explored methods for preventing or exploiting redundancies in the plans of a single agent, either during the actual planning process itself or when integrating separate sub-plans. One commonly-used technique in classical planning, *step-reuse*, tries to achieve goal conditions by preferentially introducing causal links between the effects of existing plan steps and unachieved goal conditions. This strategy avoids introducing unnecessary plan steps into a plan.

More recent work by Yang [18] has explored problems in which an agent has constructed several independent plans for separate subgoals, and now must form a single plan by merging the plans together. Yang's formal definition of plan merging states that a set (subject to some grouping restrictions) of plan steps Σ is *merge-able* with a plan step μ (meaning μ can replace Σ in the plan) if the union of preconditions of Σ subsume those of μ, the postconditions of μ subsume the *useful effects* of Σ, and the cost of μ is less than the cost of Σ. Yang states that a group of plan steps' *useful effects* are the effects that establish conditions that are preconditions of other plan steps in the overall plan. Yang's definition is flexible, in that it allows for any single plan step in a partial order plan to merge with any possible subset. However, Yang's merging criterion suffers because there are not always intuitive groupings of plan steps in an arbitrary partial-order plan, and that the number of possible groupings is quite large, making his optimal merging algorithm complex, and his approximate algorithm incomplete.

Horty and Pollack's [12] research claims that an agent should not evaluate the cost of taking on some new task by weighing the cost of the task's plan in isolation, but by taking into account how the plan meshes with its existing plans. Since the cost of a plan in context is often less than it would be otherwise (thanks to plan step merges that "kill two birds with one stone"), an agent may be able to adopt a new task without incurring an unacceptable additional cost of execution. However, Horty and Pollack only allow steps to merge when they are the same action, disallowing different actions with the same effects to merge.

Others have looked not at the single-agent plan merging problem but at ways of coordinating the plans of multiple agents in a multiagent domain [6, 11, 17]. Georgeff's coordination mechanism for agents acting in a shared environment imposes synchronization constraints between agents' actions, to guarantee that their combined execution does not cause conflicts [10]. An key element of Georgeff's strategy was to identify, through exhaustive pairwise comparisons, the subset of actions that could conflict, and grouping these together into *critical regions* to reduce the combinatorics of synchronization. More recently, Clement [2] has examined this same problem of discovering and avoiding potential inter-agent conflicts. He has shown how the plan hierarchies of the individual agents can be exploited to quickly prune uninvolved portions of agents' plans away, and to identify natural groupings of related plan steps, resulting in an efficiency improvement over Georgeff's work. We borrow heavily from Clement's ideas about the use of plan hierarchies, but focus instead on finding and exploiting positive interactions, rather than conflicts as he had done.

Multiagent techniques for coordinating positive interactions include Partial Global Planning (PGP) [8] and Generalized Partial Global Planning (GPGP) [5], which are also concerned with practical issues in avoiding exchanges and reasoning about irrelevant information. Subsequent work on space and time abstraction as a method of coordinating agents at abstract levels extends these ideas [7]. However, none of these efforts deal with explicit subsumption relationships between planning operators in agents' plans.

Ephrati has extended Yang's work on single agent plan merging to the multi-agent context by farming out subgoals of a single goal to different agents to solve and then integrate their subplans into a joint, multi-agent plan [9]. Ephrati's system was able to handle both positive and negative interactions between these subplans, but his approach suffers from similar issues of search complexity and agent commitment that Yang's does.

Recently, De Weerdt and Witteveen [3] have developed an algorithm for performing plan merging based on underutilization of free resources. Their method takes advantage of a rich plan and resource representation, but is limited to identifying and exploiting merges between grounded plans, and does not benefit from any kind of hierarchical abstraction.

The rich literatures on plan merging, hierarchical planning, and multiagent plan coordination contribute valuable components to our approach. The synthesis of these components into our new approach to finding and exploiting multiagent plan synergies reaps the combination of benefits as we seek to more efficiently search a richer space of synergies than individual prior techniques.

3. DISCOVERING HIERARCHICAL PLAN SYNERGY

In this section, we first characterize our plan representation and the summary information calculation process we rely on, then describe the synergy criteria we use, and finally, describe our search algorithm capable of discovering and exploiting these synergies. Unlike previous efforts, our research offers an algorithmic solution to the problem of merging plan steps based on subsumption relationships between the plan steps that satisfy both our desires for greater efficiency and for discovery of merges between groups of plan steps.

We point out that our approach can be used in either a centralized or decentralized manner, depending on the underlying organizational structure of the participating agents (similar to the use of Meta-level Organizations in Partial Global Planning [8]). If the agents are organized centrally

around a single authoritative agent, this single agent can aggregate the plan information and use our algorithm to reason about all possible synergy relationships between the overall group of agents. In a more decentralized organization, individual agents can share plan information with each other, and each can use our algorithm to discover synergy relationships between itself and other agents. In addition, our algorithm functions irrespective of the degree of information shared. Specifically, agents can share all or just part of their plan hierarchies (giving them the opportunity to make tradeoffs between increased efficiency and privacy), and our algorithm will use as much information as is available to discover opportunities for synergy. Unlike a conflict resolution system, there is no true performance degradation if agents do not reveal their entire plans. Rather, it is simply less likely that synergies will be found. Finally, the algorithm does not impose any single "solution" on the agents, but instead iteratively returns alternative plan merges to the agents as it discovers them. This allows the agents to use whatever means they wish to reach agreement on which solution to adopt.

3.1 Hierarchical Plan Representation

Briefly, a plan hierarchy is a hierarchy comprised of individual plan steps; at the more abstract levels of the hierarchy, each plan step achieves more effects than a plan step at a lower level, and the bottom of the hierarchy is composed of primitive plan steps that are the operators that an agent can directly execute. In essence, a plan hierarchy represents a set of possible plans that all achieve the same overall goal. Each possible complete refinement of a hierarchical plan represents one possible sequence of primitive plan steps capable of achieving the overall goal of the hierarchical plan.

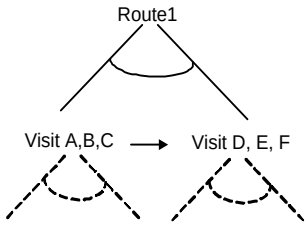


Figure 1: Agent Scout1 Hierarchical Plan

Figure 1 shows a hierarchical plan from our helicopter domain, mentioned earlier. Agent Scout1's plan is to first scout regions A, B, and C in one sector, and then scout regions D, E, and F in another sector. $VisitA, B, C$ and $VisitD, E, F$ are abstract plan steps that refine to sets of primitive actions that, together, achieve Scout1's goal of visiting all six regions.

More formally, we define a hierarchical plan step p (an element, or node, of a plan hierarchy) as a tuple {*pre*, *in*, *post*, *type*, *subplans*, *order*, *cost*, *duration*}. *pre*, *in*, and *post* are sets of conditions corresponding to the preconditions, inconditions, and postconditions of the plan step. Preconditions are conditions that must hold immediately prior to the plan step's execution, inconditions are conditions that must or may hold during execution, and postconditions are conditions that will hold immediately after execution (also known as the effects of the plan step). *Subplans* is a set of pointers to the sub-plan steps of the plan step. The type of plan step p, $type(p)$, has one of three values: *primitive*, *and*, or *or*. An *and* plan step is a non-primitive plan step that is accomplished by carrying out all of its *subplans*, and an *or* plan step is a non-primitive plan step that is accomplished by carrying out any one of its *subplans*. A *primitive* plan step has no *subplans*. *Order* is a set of temporal ordering constraints between the start or end time points of plan steps in *subplans* [16]. Constraints can be of forms $before(a, x, b, y)$, $after(a, x, b, y)$, and $same(a, x, b, y)$ where a and b are plan steps in *subplans*, and x and y are either *start* or *end* (indicating whether the constraint is between the start point or the end point of the plan step). Finally, *cost* is a real number representing the cost of executing the plan step, and duration is a real number representing the time it will take the agent to execute its plan step. For a non-primitive plan step p, the *pre*, *in*, *post*, *duration* and *cost* values can be derived from p's primitive *subplans*. A specific method for deriving this information is described in the next subsection.

3.2 Plan Condition Summarization

Reasoning about relations between abstract plan steps between different agents is more difficult than reasoning between primitive plan steps. This is the case because, unlike primitive steps, abstract plan steps do not have explicit condition information. This makes reasoning about relationships between abstract steps difficult, as it is hard to determine the exact effects of a given abstract step. For example, consider the plans of Scout1 (Route1) and Scout2 (Route2) pictured in Figure 2. To know whether the effects of plan step $VisitD, E, F$ subsume $VisitE, F$ (as is obvious based on a typical interpretation of the plan names), a merging algorithm must have knowledge about the higher-level effects of abstract plan steps, and not just the effects of the primitive plan steps that compose the abstract step.

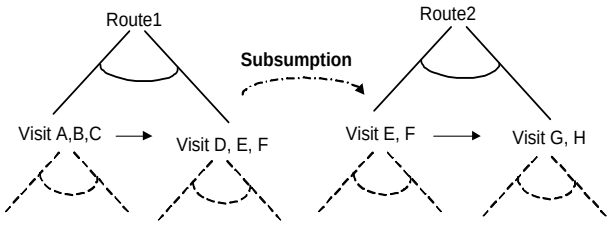


Figure 2: Scout1 and Scout2 Hierarchical Plans

To solve this problem, we use an existing algorithm built by Clement [2] that propagates such condition information up the plan hierarchy, so that abstract steps derive their precondition and postcondition information from their subplans. Clement has used this technique to help resolve negative interactions between the hierarchical plans of multiple agents at various levels of abstraction [2]. In our work, we also recognize and exploit the fact that this summary information can also be used to identify subsumption relationships between abstract plan steps of different agents.

Clement propagates condition information from the leaves of the hierarchy to the root plan step [2]. To represent this summary information in the plan hierarchy once it is calculated, Clement makes a modification to the hierarchical plan tuple we described previously. A plan condition c ($c \in pre$, *in* or *post*) is now represented as a tuple, $\{l, existence\}$. l

is the condition literal, the actual condition that will hold, and $l(c)$ refers to the literal of condition c. The existence of c can be *must* or *may*. If it is *must*, then c is called a *must* condition because l holds for every successful plan execution. If c is *may*, it is called a *may* condition because it will hold for some possible executions, but is not guaranteed to hold for every successful plan execution. This introduction of *must* and *may* conditions is necessary as, unlike a primitive plan step whose conditions are always necessary (i.e., always *musts*), an *or* abstract plan step will involve different conditions depending on how it is refined.

3.3 The Top-Down Search Mechanism

3.3.1 Plan Step Merging Criteria

To discover possible plan step merges based on plan step condition subsumption relationships, we have constructed an algorithm (henceforth referred to as the Synergy Algorithm, or SA) capable of finding and implementing plan step merges between different agents' hierarchical plans. This algorithm is modeled after Clement's algorithm in [2], though we extend it in key ways that set our research apart from his work. Our algorithm identifies pairs of plan steps that share a subsumption relationship, and merges them by removing the subsumed plan step and imposing additional constraints to ensure correct execution. Though our algorithm currently finds only pairwise merges, if three or more agents can merge plan steps such that only one of them needs to execute a plan step to satisfy them all, the SA can find such a merge through repeated discovery and adoption of pairwise merges in the same run of the algorithm. This allows the algorithm to coordinate any number of agents at a time, though it does not support all types of plan step reallocations [15].

The SA determines if any two plan steps can merge by first calculating the summary information for each of the hierarchical plans using Clement's techniques [2], or rely on the agents to perform this calculation themselves, before submitting their plans to the agent(s) running the SA. The SA then examines the summarized postconditions of the two plan steps. If one of the two plan steps has a set of summarized *must* postconditions that subsumes the summarized *must* postconditions of the other plan step, then they can be merged. Using Yang's terminology, we say that the step with the subsuming postconditions is the *merged* step [18].

More formally, we say that plan step p_i can be the merged step of another plan step p_j (meaning p_j replaces p_i in the new plan) if and only if

$$\forall c_i, c_i \in post(p_i) \wedge existence(c_i) = must$$
$$\Rightarrow$$
$$\exists c_j, c_j \in post(p_j) \wedge existence(p_j) = must \wedge l(c_i) = l(c_j)$$

This is a modified formulation of Yang's original criteria for plan step merging [18]. Yang's criteria additionally require that $pre(p_i) \subset pre(p_j)$. We have relaxed this restriction. Since the existing plans are assumed to be correct, p_j does not have to rely on the steps that were achieving conditions to enable p_i to execute. The removal of a plan step p_i may make other plan steps unnecessary, allowing them to be dropped as well. Our mechanism currently does not support the removal of these auxiliary actions, though we address this issue in our future work section.

Intuitively, the SA should be able to merge on *may* conditions as well as *musts*. By definition, *may* conditions at a higher level of abstraction are *must* conditions at a lower level of abstraction. Hence, to take advantage of possible merges between *may* conditions, the SA will encounter these upon further decomposing the hierarchy (by making selections at *or* branches) to the point that the *may* conditions become *must*, and merges them at that point.

3.3.2 Algorithm Description

The search for plan merges is through the space of partially expanded agent plan hierarchies. We term these partial expansions *frontiers*; the search can thus also be characterized as a search through the space of possible agent plan frontiers.

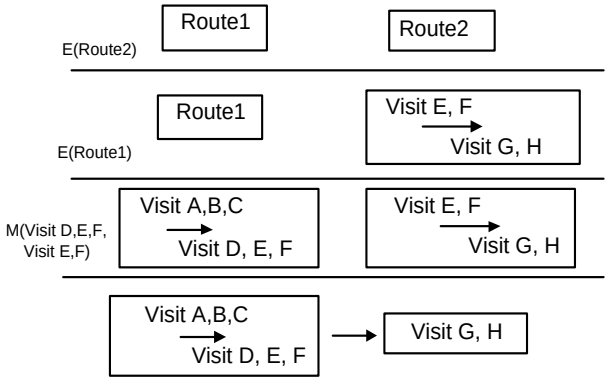


Figure 3: Algorithm State Expansion

Our SA implements a top-down search, because the algorithm first explores the abstract levels of the agents' hierarchies before expanding them to more primitive levels. This means that the first plan step merges that are discovered are between plan steps at an more abstract levels within the agents' hierarchies. The SA starts the search at a state containing the top-level frontiers of all agents being coordinated and expands downwards, iteratively returning states in which new merges have been performed. States on the search queue are ranked based on their overall cost, where the overall cost of a search state is calculated by summing the individual costs of the plan steps on each agent's plan frontier. As it finds states that have lower overall cost than previously seen states, it returns them as candidate solutions to the agents.

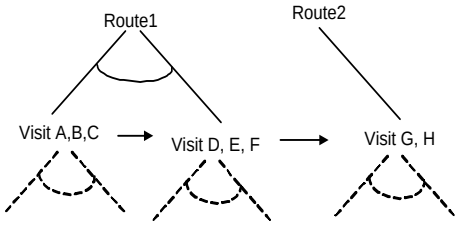


Figure 4: Scout1 and Scout2 Merged Plan Hierarchies

Figure 3 shows a series of expansions of the helicopter agents' plan hierarchies, representing one path through the search space of frontiers. In the figure, the algorithm starts at a state with the top-level frontiers of both agents (the roots of their hierarchies) and expands down. It discovers

that plan steps $VisitD, E, F$ and $VisitE, F$ can be merged, and merges them to produce a new search state. Additionally, an ordering constraint is added between the two plan steps to be consistent with the ordering that Scout2 had between its original actions. Figure 4 shows the solution that would be added to the solutions list upon reaching this state.

```
Begin Synergy Loop
    If SearchQueue empty, exit
    s = Pop(SearchQueue)
    PruneFrontiers(s)
    for each x in Frontiers, y in Frontiers
    {
            if postcondSubsume(x, y)
                s2 = MergePlans(x, y, s)
            else if poscondSubsume(y, x)
                s2 = MergePlans(y, x, s)
            if(!CycleDetect(s2))
                PriorityInsert(s2, SearchQueue)
    }
    for each x in Frontiers
        if !(x.primitive) & !(x.pruned)
    {
        s2 = ExpandPlan (x, s)
        Insert(s2, SearchQueue)
    }
    if(Deconflict(s, solutions_list))
        Insert(s, Solutions)
End Loop

MergePlans (a,b,s){
    s2 = Copy(s)
    Remove(b, s)
    UpdatePartialOrderings(a, b, s)
    Return s2   }
ExpandPlan (a, s){
    s2 = Copy(s)
    Remove(a, s)
    Insert(a.subplans, s)
    SubstPartialOrderings(a, a.subplans, s)
    Return s2   }
```

Figure 5: Synergy Algorithm

Figure 5 outlines our Synergy Algorithm. A state in our algorithm is a tuple $\{frontiers, orderings\}$. $frontiers$ is the current set of plan frontiers of the planning agents, and $orderings$ is the set of inter-agent and intra-agent ordering constraints added by merging and expanding plans. The algorithm begins by de-queueing the first search state from the search queue. If the de-queued state is already on the closed list of states, it has already been expanded, and so is discarded and a new state is de-queued. The process stops when there are no longer any states remaining on the queue to be expanded.

If the dequeued state is not on the closed list, it is added to the closed list and the algorithm generates successor states created by merging plans. The MergePlans function tests search states to determine if plan steps on one agent's frontier in $frontiers$ could pairwise merge with plan steps on any other agent's frontier in $frontiers$. For each pair of plan steps that can merge, the SA generates a new search state in which the plan step that was subsumed is removed from its frontier. After the merge, if the state is cheaper than any previous candidate solution and is found to have resolvable conflicts (more on this below), the algorithm adds the state to its list of solutions.

The algorithm also generates successor states by expanding existing plan steps. For each plan step of type *and* on each frontier, it generates an additional search state in which the plan step has been replaced on its frontier by its subplans. For each plan step of type *or* on either frontier, the algorithm generates multiple successor states by replacing the plan steps by each of its subplans. The expansion of *or* plan steps allows for potential merging of children of an *or* plan step by committing the executing agent to a particular subplan of that *or* plan.

A key to understanding our search process is to understand that the iterative discovery of plan steps to merge between agents' frontiers is integrated in the actual search process. That is, new states in the search are generated both by expanding existing plan steps on the frontiers and by merging two plan steps to generate a new state with frontiers created by the merge of two plan steps.

3.4 Plan Frontier Pruning

The space of possible frontiers across all agents is exceedingly large. The number of frontiers possible in a single, balanced hierarchy with a uniform branching factor B and all *and* plan steps can be calculated using the following recursion, where d is the depth of the tree:

$$Frontiers(d) = Frontiers(d-1)^B + 1 \qquad (1)$$

Thus, the worst-case complexity of the search space of possible combinations of agent plan frontiers is quite large. Clearly, any search algorithm exploring this space (like the search our SA implements) would like to avoid generating all possible frontiers as it searches for plan step merges between agents' hierarchies.

To address this problem, we use frontier pruning to avoid the unnecessary generation of search states. Before a dequeued state generates its successor states based on the contents of its frontiers, the SA checks each plan step on one frontier against the plan steps on the other agent's frontier to determine if any of the plan step's postconditions unify with those of any plan step on the other frontier. Plan steps that have no overlapping postconditions are marked as *pruned*, meaning that the plan steps are never expanded to generate new search states, since a plan step with no overlapping postconditions will not have any children with postconditions that overlap either.

In general, we assume that the SA will be applied in cases where agents are acting nearly independently. Nearly independent agents' plan hierarchies will predominantly affect different aspects of the world, and thus will have many pruned plan steps. This allows the SA to find the few actual merges much more quickly than it could otherwise. This feature can significantly reduce the search time in problems where there is little potential for synergy, as it reduces the number of possible successor states that could be generated from a state. This process is similar to one used by Clement [2], though while he prunes plan steps based on a lack of conflict, we prune based on a lack of synergy.

3.5 Implementing Plan Merges

Once the SA has identified a pair of plan steps that can merge, the solution it proposes must provide additional information besides specifying which agent should drop a par-

ticular (subsumed) plan step from its plan. The solution should also constrain the involved agents to only executing plans that can be refined based on the particular frontiers for which the solution has been proposed, rather than from any plans that could be refined from the root plan steps of their original hierarchies. The following subsections detail the other specifications that are essential for ensuring correct execution after adopting a merge.

3.5.1 Maintaining Partial Ordering Correctness

Each time the SA merges two plan steps between different agents' hierarchies, it must modify the partial orderings over agents' plan steps in the frontier to implement this merge and ensure that dependencies on the plan step being removed are replaced with dependencies on the plan replacing it. To modify the ordering constraints after merging two steps a and b (where b is taking the place of a in the plan), the SA must carry out three steps (Figure 6 illustrates this concept graphically): first, the SA removes all ordering con-

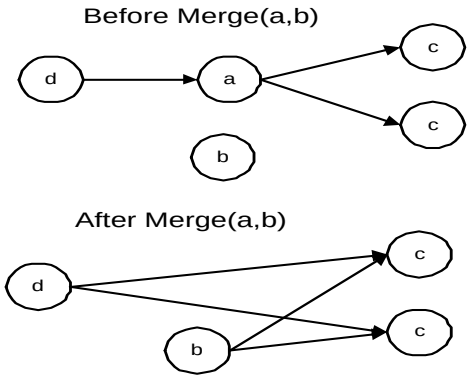


Figure 6: Updating Temporal Constraints After a Merge

straints between the plan step a being removed and another plan step c of form before(a, c) (or after(c, a)) from the constraints list. Second, for each of the previously removed constraints, a new constraint is added between c and the plan step b replacing the removed plan of form before(b, c). Finally, for all plans in constraints of form before(d, a) (or after(a, d)) and points c (defined above), we add a constraint of form after(c, d). This ensures that all existing orderings are preserved. This functionality for determining orderings is implemented as the UpdatePartialOrderings function in the MergePlans function in Figure 5.

To maintain temporal consistency, the SA also modifies partial orderings when it expands plan steps in search states. When a child or the children of a plan step replace it on a frontier, all constraints between the original plan steps and other plan steps on either frontier are removed by the SA, and it creates new constraints between these other plan steps and the original plan step's children, such that the existing temporal orderings are ensured.

Once the SA has implemented either of these ordering constraint changes, it determines if the transitive closure of the partial orderings between the plans in both frontiers contains any cycles and thus must be rejected as a potential merge or expansion. The the SubstPartialOrderings function in the ExpandPlan function in Figure 5 implements this functionality.

3.5.2 Resolving Planning Conflicts

To identify and resolve conflicts, we incorporate a conflict resolution method developed by Clement [2]. His method identifies plan steps in one agent's frontier that potentially threaten conditions associated with plan steps of another agent's frontier, and either ensures that ordering constraints are added to prevent this threatening step from clobbering any conditions, or if this is not possible, rejects the current solution candidate. As the SA expands the agents' frontiers and generates potential merged solutions, it first passes the search state through this plan conflict resolution mechanism, before it enqueues the state on its candidate solutions list.

4. EVALUATION

Recall that our goals for an effective plan merging algorithm were, first, that it should find plan step merges efficiently, and second, that it should look more broadly for synergies between groups of (primitive) plan steps, and not just at replacing plan steps one-for-one or one-for-many. In this section, we provide experimental results to justify that our top-down approach to hierarchical plan merging achieves both of these goals.

In order to quantify the efficiency advantages of our top-down approach, we conducted two sets of experiments in which we compared it to a baseline algorithm that simply produces all combinations of fully expanded plans for all agents and looks for plan step merges just between primitive plan steps. Our comparison assumes hierarchical information is available, and does not consider the costs in constructing hierarchies where none previously exist, because of the prevalence of hierarchical planning knowledge in the types of applications we study. For both sets of experiments, we assume that the plans have already been summarized offline.

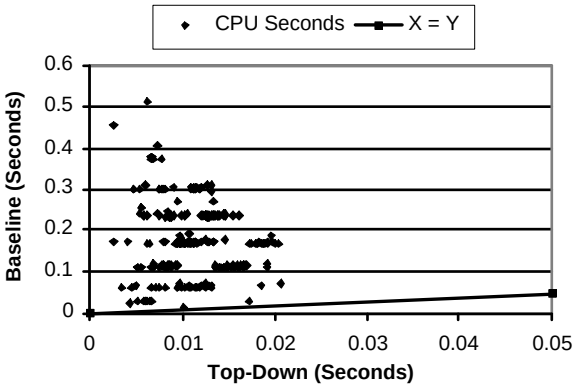


Figure 7: Top-down Search vs. Baseline Search

The first set of experiments is an efficiency comparison between our algorithm and the baseline algorithm, comparing how fast each algorithm discovers the cheapest between two plan hierarchies. Both plan hierarchies were *and* plan hierarchies of depth three and branching factor two, where each primitive had a single postcondition associated with it (giving each hierarchy eight postconditions). We gave the first agent the same hierarchy each time and then tried all variations on the number of overlapping postconditions (but

not the location of conditions that overlap) between the first agent's hierarchy and the second agent's hierarchy. (For example, if the first agent's hierarchy was composed of the conditions ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h') then one possible hierarchy for the second agent would be ('a', 'x', 'y', 'd', 'z', 'f', 'g', 'h')). Figure 7 plots each of the 256 experiments based on the CPU time required by each algorithm to find the (same) best merging solution. In all cases, the CPU time for the baseline algorithm is at least that of the improved algorithm (note the different scale on the graph) and in most cases, much greater.

Though these efficiency results correspond only to the two-agent synergy case, they should apply equally to the case when we coordinate three or more agents. Though the complexity of the top-down search increases as more agents are added (mainly because there are more abstract plan steps to compare for possible synergies), the complexity of the primitive-only search grows even more, because of the exponentially larger number of primitive plan steps than abstract plan steps.

Merging abstract plan steps is advantageous for heterogeneous agents who have different primitives for accomplishing actions, but whose abstract plans accomplish the same (or subsumed) effects. In this case, none of their primitive steps could merge, but the abstract steps that combine primitives can. The merging of two abstract steps is effectively a many-to-many merge between two different groups of plan steps, as a single abstract step represents a group of primitive plan steps. This is a capability not present in Yang's algorithm nor in the baseline algorithm from above.

Both the baseline algorithm and Yang's optimal plan step merging algorithm *can* merge a single step with a group of steps, where the single step's conditions subsume those of the group. The baseline system will achieve this by performing a series of pairwise merges between the subsuming step and each step of the set of steps being subsumed. However, even for these merges, both approaches suffer from the problem of not having an intuitive sense of what sets of primitives to group together. Given n primitives at the bottom of a plan hierarchy, the number of possible subsets of primitive plans is

$$\sum_{k=1}^{n} n!/((n-k)!k!) = 2^n$$

Yang's complexity analysis of his algorithm reveals the same exponential result [18]. Yang does present an $O(n \log(n))$ approximation algorithm for performing plan step merging, though this algorithm relies on a total order over the individual plans, which is an assumption not made of the plans handled by our algorithm.

In contrast, our top-down approach relies on the pre-existing structure and organization of primitives into more abstract tasks (just one of many possible subsets), and thus can avoid complexity issues if it is able to identify merges between abstract steps. Additionally, in the worst case, when the primitives in the hierarchies are not grouped well into abstract steps that produce synergy at the abstract level, our SA will still identify merges between a single subsuming plan step and a group of plan steps in the same manner as the baseline approach does.

To show how our SA is able to find merges that the baseline approach cannot, we conducted experiments that compared the SA's and the baseline algorithm's performance on

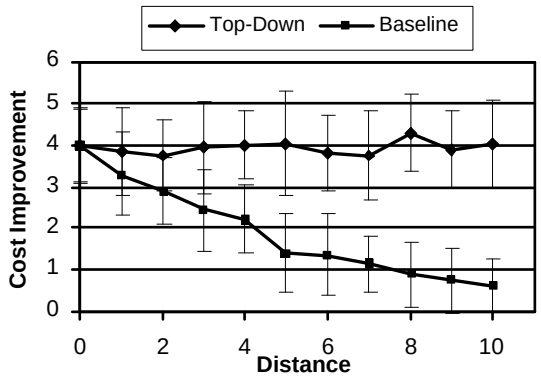


Figure 8: Comparing Cost Reduction

sample plan hierarchies in which the agents shared common effects but varying degrees of difference in how their primitives achieve the effects. Each experiment involved a search for synergy between two agents' hierarchies, in which we randomly generated the first agent's hierarchy with a depth of four and branching factor of two with two postconditions at each primitive, meaning that the hierarchy had sixteen actions with a total of 32 postconditions. All subplans were totally ordered. We then created the second agent's hierarchy by shuffling some of the common postconditions between the primitive plan steps. For each experiment, the plans had half (sixteen) of their conditions in common.

Figure 8 displays the results of these experiments. The X-axis represents the *distance* of one hierarchy from the other, where distance was computed as the number of common postconditions that were located in different primitives. The two lines graphed represent the average cost improvement (over twenty runs, with error bars indicating one standard deviation on each side) of both our SA and the Baseline algorithm presented earlier. The graph shows that, as the distance between hierarchies (the agent heterogeneity) increases, the top-down search is increasingly able to find solutions between abstract and primitive plan steps that the Baseline algorithm misses. Thus, even in the simplistic random plan hierarchies used here, we can demonstrate substantial improvements due to our new top-down algorithm. Our ongoing research involves a broader investigation into the characteristics of more complex plan hierarchies that influence heterogeneity, and how those characteristics differentially impact the efficacy of our approach along various measures of performance.

Another benefit from our top-down approach is that it represents an incremental search (an "anytime algorithm" [4] for synergies. Typically, the more time spent, the more deeply into the plan hierarchies the synergy search goes, and the greater the opportunity for breaking plans into finer pieces that can be merged to more precisely balance the plans' costs among the agents. Such a balance can, for example, increase parallelism to decrease the plans' durations. This suggests that there is a "break-even" point in using our top-down synergy search - a point at which the synergy improvements (savings in plan execution time) stop exceeding the time spent finding those improvements. We can empirically see this, as illustrated in Figure 9. An advantage of our

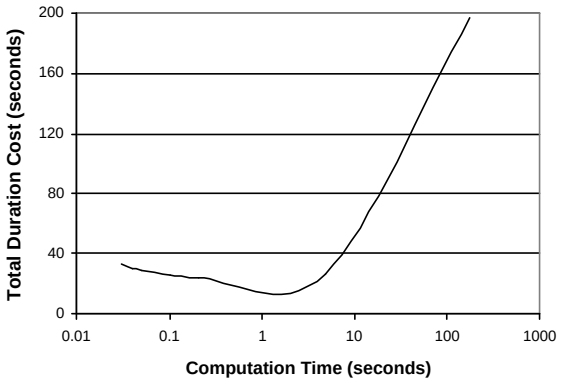


Figure 9: Tradeoffs of coordinating at different levels

approach is therefore its "anytime" character, allowing the use of deliberation scheduling [1] or other methods to temper the search for synergies based on its costs and benefits.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we described a method of searching for and exploiting synergistic relationships between different hierarchical planning agents in a multi-agent system. We have shown how an effective search strategy (a top-down search) combined with a plan summarization mechanism, which have proven effective for conflict resolution [2], can also benefit a search for these synergies. Our experimental results indicate the effectiveness of our approach when compared to a system that does not harness the advantages of a hierarchical plan representation.

We have also been working on ways to add reasoning about causal links between plan steps in plan hierarchies to enable us to automatically determine the *useful effects* [18] of plan steps, which in turn will allow us to determine what remaining steps in a plan frontier are still serving a useful purpose. Those that no longer contribute conditions to later steps (as causal links indicate) can be removed from the plan hierarchies. That is, steps that do not have any causal links connecting their postconditions to another step's preconditions can be dropped as well. Our most current work implements the use of causal links to determine useful effects. In addition to our causal link work, we are also working to enhance our representation to support both temporally quantified relations and constraints between plan steps, as well as expanding the SA to discover and exploit additional types of synergy relationships beyond the effect subsumption relationships considered here.

6. ACKNOWLEDGEMENTS

We wish to thank Brad Clement, Thom Bartold, and the CoABS Grid development team for software they contributed to this effort. This research was supported by DARPA (F30602-98-2-0142).

7. REFERENCES

[1] M. Boddy and T. L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[2] B. Clement and E. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. of Third Int. Conf. on Autonomous Agents*, pages 252–259, 1999.

[3] M. M. de Weerdt and C. Witteveen. A resource logic for multi-agent plan merging. In *Proc. of the 20th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 244–256, 2001.

[4] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of AAAI-88*, pages 49–54, 1988.

[5] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *Proc. of First Int. Conf. on Multi-Agent Systems*, pages 73–80, 1995.

[6] M. desJardins and M. Wolverton. Coordinating planning activity and information flow in a distributed planning system. In *AAAI Fall Symposium on Distributed Continual Planning*, 1998.

[7] E. Durfee and T. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1363–1378, 1991.

[8] E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, Sept.-Oct. 1991.

[9] E. Ephrati and J. S. Rosenschein. Divide and conquer in multi-agent planning. In *National Conference on Artificial Intelligence*, pages 375–380, 1994.

[10] M. Georgeff. Communication and interaction in multi-agent planning systems. In *Proc. of AAAI-83*, pages 125–129, 1983.

[11] C. V. Goldman and J. S. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proc of the 12th Int. Workshop on Distributed AI*, pages 171–185, 1993.

[12] J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, 2001.

[13] A. L. Lansky. Localized search for controlling automated reasoning. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 115–125. Morgan Kaufmann, 1990.

[14] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[15] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transaction on Computers*, number 12 in C-29, pages 1104–1113, 1980.

[16] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. of Fifth National Conf. on AI*, pages 377–382, 1986.

[17] F. von Martial. Interactions among autonomous planning agents. In Y. Demazeau and J.-P. Muller, editors, *Decentralized AI*, pages 105–119. North Holland, 1990.

[18] Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach to Classical Planning*. Springer-Verlag, Berlin, 1997.

Appendix H:

Limiting Disruption in Multiagent Replanning

Thomas Bartold
University of Michigan
1101 Beal Ave
Ann Arbor, MI 48109-2110 USA
1-734-763-9074
tbartold@umich.edu

Edmund Durfee
University of Michigan
1101 Beal Ave
Ann Arbor, MI 48109-2110 USA
1-734-936-1563
durfee@umich.edu

ABSTRACT
Multiagent systems sometimes undergo changes that cause coordination commitments to become insufficient or out of date, such that the coordinated agent plans need to be repaired or replaced. When recoordination becomes necessary, disruption to the commitments made by the agents in their original plans should be minimized. We approach the problem of minimizing disruption by augmenting pre-existing coordination technology by developing metrics and automated processes for it to rank and potentially recommend new coordination commitments more rapidly. In this paper, we explain, examine, and evaluate our new metrics and processes, demonstrating empirically that flexible measures of disruption can streamline the coordination process.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence - *Multiagent systems*

Keywords
Coordination of multiple agents, Conflict resolution, Multiagent planning, Biased search, Disruption

1. INTRODUCTION
Agents acting in complex, dynamic environments must often adjust their plans to take into account actions of other agents, such as to avoid potential conflicts. Possible conflicts can be detected by selectively exchanging and comparing portions of agents' individual plans, identifying inconsistent expectations, and adding synchronization actions and/or blocking some action choices to ensure conflicts cannot arise. Reaching agreement on what coordination commitments to adopt, and implementing them (which could trigger further downstream commitments and negotiations), can incur significant overhead. For example, in one of the application domains that we have studied, reaching commitments for multi-national coalition operations [13] can involve numerous levels of negotiation and diplomacy.

AAMAS'03, July 14-18, 2003, Melbourne, Australia.

If some of the agents need to later revise their individual plans in response to unexpected dynamic changes in the environment, prior coordination commitments can be imperiled. To avoid having to incur another round of significant overhead, the agents might prefer to coordinate their changed plans as similarly as possible to how they coordinated their previous plans. We refer to changes in coordination commitments as forms of **disruption.**

We argue that agents should consider disruption when deciding on coordination commitments to make in changing circumstances. Obviously, there are many factors besides disruption that should be weighed when deciding on a coordination solution, including reducing costs for joint activities and maximizing parallel and independent activity of agents. However, the emphasis of this paper is on disruption. In particular, the contributions of the work we present in this paper revolve around how disruption can be estimated when comparing alternative coordination solutions, and how the search for coordinated solutions can be effectively streamlined towards finding less disruptive candidates sooner.

In what follows, we assume that the plans of a group of agents have been initially coordinated, but that a change occurs that forces one or more agents to revise their plans such that some coordination commitments must be violated. Rather than starting the planning process over again from scratch, or even just starting the coordination process over again, we want to reuse the results of the prior planning and coordination process to guide the search to restore coordination in a manner that minimizes disruption to the previous coordination commitments, effectively repairing the coordinated solution to fit the revised plans.

The idea of attempting to reuse prior planning and coordination effort when adapting to new situations is far from new. Indeed, issues of how and when an agent should decide whether to honor its prior plan commitments versus reconsidering them have been at the core of research in areas such as plan repair/replanning [8], reactive plan execution architectures [6], and plan management [7]. Similar issues arise in the multiagent literature, in terms of how an agent that has already coordinated its plans with other agents should react to unexpected circumstances. Relevant work includes that on conventions that agents abide by when they change their intentions [5], on *polite* behavior that attempts to minimize impact on others [11], and on tolerating minor inefficiencies and adhering to suboptimal plans rather than prompting recoordination [4]. Our work adds to this thread of research by developing some rudimentary metrics for particular kinds of disruption, and by demonstrating how such metrics can be heuristically employed to improve coordination search efficiency. In contrast with PGP [4], where inconsistency implied inefficiency, here any inconsistency could be catastrophic, and

recoordination becomes imperative. Because the need for recoordination can be time sensitive, coordination efficiency is an issue, and could benefit from prioritizing the order in which conflicts are addressed, as in constraint satisfaction problems [14].

In the next section, we illustrate concepts of disruption using a simplified application domain that we later elaborate for experimental evaluation. We then turn to metrics for estimating disruption, which forms the basis for comparing alternative candidate solutions. Given a search mechanism that generates candidate solutions, we can rank these solutions based on the metrics. However, given the assumed dynamics of the environment, it could be the case that candidate solutions generated later are better, and it would be desirable instead to bias the search process to generate less disruptive solutions sooner. We outline our techniques for instituting this bias in Sections 4-5. In Section 6, we evaluate the effectiveness of these techniques, and then we outline directions of our future work in the concluding section.

2. SIMPLIFIED EXAMPLE

To date, the primary application domain in which we have studied disruption has been coalition operations, where the plans of coalition partners might unintentionally interact, and so the partners coordinate their plans, and recoordinate their plans, only when circumstances change [13]. For ease of exposition and experimentation, however, in this paper we will concentrate on a simplified problem of coordinated access to potentially shared resources, in terms of avoiding collisions of aircraft that could share corridors. The alternative routes and locations are represented as edges and nodes (respectively) in a graph, as in Figure 1. The first agent, A_1, wants to move from location A to location D, and can do so either via location B or via location C.

Our research has emphasized coordinating plans generated through HTN planning techniques, in the tradition of NOAH [9] and NONLIN [10]. Planning begins with high level goals and iteratively refines abstract steps until a complete task network is created. The plan hierarchies generated are all strictly trees. By retaining the hierarchy, coordination can be achieved at various levels of abstraction. In Figure 2, we show how A_1's plan could be represented hierarchically [8,12] with three abstract plan steps and four primitives. For A_1, the "Cross AD" abstract plan can be refined to either abstract subplan "Via B" or "Via C". Each of these subplans can be refined into a pair of primitive move operations. The plans used in our example are assumed to be totally ordered, although this is not required by our methods.

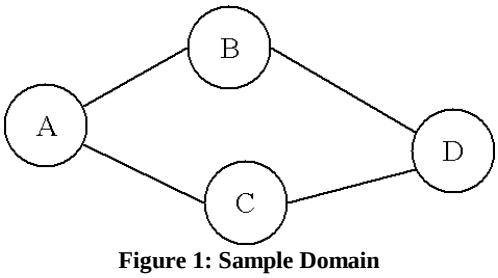


Figure 1: Sample Domain

An extreme example of overlap is when a second agent, A_2, also needs to go from A to D, and has access to the same routes. It therefore has an identical plan to A_1. In this case, the agents could coordinate by synchronizing their actions, such as by having A_2 wait to begin its movements until it receives a signal from A_1, and A_1 sending that signal once it reaches D. Alternatively, the agents could impose constraints on how each will execute its plan, such as by having A_1 commit to not move through location C, and A_2 commit to not moving through B. These (and other possible) coordination solutions impose constraints on what agents are allowed to do, what they are required to do, and/or when they can do various actions.

Assuming that agents have agreed on particular commitments and have gone through the (often considerable) effort to reserve the resources and institute the agreements to implement them, then they should resist making wholesale changes if the world changes. For example, a third agent, A_3, could enter the picture; in the worst case, its plan might be the same as those of A_1 and A_2. The simplest, and least disruptive way, of coordinating A_3 would be to simply force it to work around the coordinated plans of the other agents. For example, it might simply wait until A_2 signals it (once A_2 is done) if A_2 was constrained to first wait for A_1. Or, if A_1 and A_2 had committed to complementary routes, then A_3 either would need to be signaled by both of them (in which case it could follow either route) or one of them (in which case it would have to traverse the same route as that one). But, given the delays of either of these, it might be preferable for A_3 to be permitted to begin as soon as one or both of the others have completed only their first legs. This imposes more demands on those other agents, and is more potentially disruptive. If A_3 had a time-critical delivery to make, then it might want to precede one or both of the other agents, which could more significantly disrupt the previously expected behaviors of $A_1$and A_2.

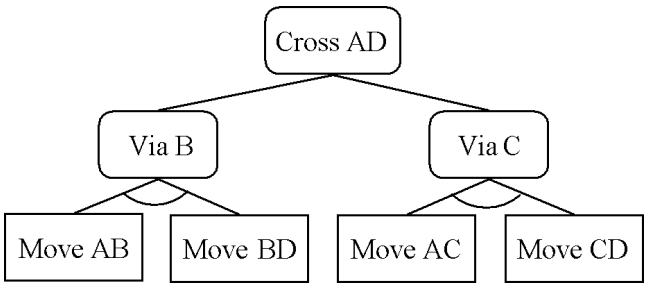


Figure 2: Sample Hierarchical Plan

Again, as illustrated in this example, our claim is not that wholesale changes should not be permitted, but rather that, all else being equal, less disruptive solutions should be preferred. In this example, and in less tightly-coupled examples (where agents have overlapping but not identical plans), we would like to have measures of how much disruption is involved. Doing so requires a clearer characterization of the types of interagent commitments that can be made, and the costs of making and breaking them.

3. COORDINATION COMMITMENTS

Coordination commitments can be viewed as modifications to single-agent plans that constrain their execution and augment their actions to ensure coordinated outcomes. We assume plans are organized hierarchically, as in our previous example illustrated in Figure 2. A plan P_i (for agent A_i) is represented as a tree where each node is a plan step that can be identified (or labeled) uniquely as accomplishing some goal or subgoal. An *and* plan step (denoted by the connecting line between its children) or an *or* plan step can be refined, replacing an *and* step with all of its

children, and replacing an *or* step with any one of its children. The pre- and post-conditions of an abstract plan step can be summarized from those of its children [1,2]. Primitive plan steps are neither *and* nor *or* because they cannot be refined, and represent actual actions to be taken by the agent, where each action has standard STRIPS representations of the action's pre- and post-conditions. Primitive plan steps also have costs and durations, where the interval over which the plan step takes place is captured as start and end time points. The preconditions, postconditions, costs, durations, and start/end time points of non-primitive plan steps can be derived bottom-up from their descendants. A plan step can also have in-conditions, permitting it to model temporary changes to the world (brought about by some of an abstract step's subsumed primitives and undone by other primitives) as well as more permanent effects of the step.

A coordination solution $C = (A, P, S, B)$ is composed of a set of agents, $A = \{A_1, A_2, \ldots, A_n\}$, where the number of agents $n = |A|$, the individual agent plans, $P = \{P_1, P_2, \ldots, P_n\}$, that are each internally conflict-free, and coordination commitments that resolve conflicts between the agent plans. There are two distinct types of coordination commitments: blocking commitments $B = \{B_1, B_2, \ldots, B_n\}$, and synchronization commitments, $S = \{S_1, S_2, \ldots, S_n\}$, partitioned into subsets associated with each agent. Blocking commitments made by an agent capture promises by the agent to block off some of its options (restrict its choices for an *or* plan step) when those options are responsible for potential conflicts. Synchronization commitments represent a decision by an agent to engage in a signaling operation with other agents, either waiting for the other agent to signal it such that it can continue with its plan, or sending a signal to another agent to release that other agent to continue.

Whenever the plans of any of the agents change in response to domain dynamics, including when agents enter or leave the domain, the agents' previously-held coordination solution, $C'=\{A', P', S', B'\}$, can become obsolete. The disruption (D) caused by a new coordination solution (C) as compared to the previous (reference) solution (C'), can be calculated solely by examining the sets of coordination commitments in the solutions $D(S', B', S, B)$. The simplest metric is to just count the number of new commitments added into the new solution and the number of old commitments dropped from the previously adopted solution. However, this unrealistically treats all commitment changes as equally disruptive. Instead, we weight changes differently depending on the commitment type, as now described.

3.1 Synchronization Commitments

To avoid conflicting actions, one agent may (at a specific point in its plan) need to notify another agent that the other agent may proceed (from a particular point in its plan). In our simple example, for instance, when A_1 has completed the action "*Move AB*", it could release A_2 to take its first moving action. Each synchronization commitment can be represented by a 5-tuple $(A_i, point_i, type, A_j, point_j)$. The *type* of synchronization commitment will be either *wait* (for another agent), or *release* (another agent to proceed). Each synchronization commitment has a dual; that is, for every *wait* there is a *release* at some other agent, and *vice versa*. These commitments always occur right before or after a particular plan step, specified for A_i as $point_i$. This means that synchronization messages always occur between (primitive) plan steps; however, they can occur during more abstract plan steps, so long as exactly when during those steps is ultimately derivable as those steps are refined. A_1's commitment above would be represented as $(A_1, move\text{-}ab\text{-}end, release, A_2, cross\text{-}ad\text{-}start)$.

We assume that disruption is additive, so that the total disruption by some coordination solution can be computed as a summation over individual disruptive changes to commitments. In what follows, therefore, we first quantify the disruption due to individual commitment changes, and then we go on to explain how we determine the disruption to the agent when there are multiple elements in one set of commitments or the other.

In our example, at the most abstract level, A_1 could wait for A_2 to complete its plan before starting its own, resulting in a single coordination commitment for each of the agents. This would be represented as the coordination commitment:

$$S_{11}' = (A_1, cross\text{-}ad\text{-}start, wait, A_2, cross\text{-}ad\text{-}end) \quad (1)$$

and its dual:

$$S_{21}' = (A_2, cross\text{-}ad\text{-}end, release, A_1, cross\text{-}ad\text{-}start). \quad (2)$$

When looking for matches, our algorithm examines the attributes of the various coordination commitments. Weighting each of these attributes appropriately can scale the quality of the match. That is, a commitment that matches on four out of five attributes should have lower disruption than one that can only be matched with a single attribute. For example, assume that given a change to the agents' plans, one possible new commitment is:

$$S_{11} = (A_1, cross\text{-}ad\text{-}start, wait, A_2, move\text{-}cd\text{-}end) \quad (3)$$

Compared to the original commitment (Eq. 1), an attribute does not match, so there is a change in commitments and this contributes to the disruption to A_1. Differences (deltas between commitments) associated with different attributes can be weighted differently. In this case, the *other agent's time point* ($point_2$) does not match, so a delta (δ) for other, δ_o should be added to the total computed disruption to A_1.

The total disruption is simply a multi-attribute summation over the deltas, where a weighting (δ value) is associated with each non-matching attribute. In the case of synchronization commitments for any agent A_i, there are four attributes that might not match[1], and their weights are given as δ_e (A_i's plan step time *point*), δ_c (the *type* of commitment), δ_a (the other *agent*), and δ_o (the other agent's plan step time *point*).

If the synchronization constraint had instead switched the order of the plan executions, then a new commitment could instead be:

$$S_{11} = (A_1, cross\text{-}ad\text{-}end, release, A_2, cross\text{-}ad\text{-}start) \quad (4)$$

Compared to the original commitment (Eq. 1), this has three attributes that differ (only the agents remain the same). The disruption would thus be the sum $\delta_e+\delta_c+\delta_o$.

In general, the disruption between a pair of commitments can be directly calculated from the two commitments by treating them as arrays, and applying the formula:

[1] Since disruption is evaluated for each agent separately, the first attribute of all constraints applied to a specific agent must match.

$$Ds_{ijk} = \delta_e(S_{ij}[1] \neq S_{ik}'[1]) + \delta_c(S_{ij}[2] \neq S_{ik}'[2]) + \delta_q(S_{ij}[3] \neq S_{ik}'[3]) + \delta_o(S_{ij}[4] \neq S_{ik}'[4]) \quad (5)$$

where $S_{ij}[0]$, the first attribute, is the agent and will be the same for any commitments being compared. Mismatches in any of the other four attributes will contribute the appropriate δs. Ds_1 is a measure of disruption to agent A_1 and the brackets indicate the indices of the synchronization constraints in the two sets of commitments.

3.1.1 Unmatched Synchronization Commitments

Not all plan commitments will match between solutions; one set of commitments may be larger than the other. A synchronization commitment from the old solution that is unmatched in the new solution could be disruptive to an agent, but the amount of disruption should depend on the *type* of commitment. If A_2 were prepared to *wait* for A_1 to release it to proceed, we would expect the absence of that signal to be disruptive. We can identify this kind of situation by noticing that a *wait* commitment has been dropped from the set of commitments for the agent. A_2 must notice that the commitment is missing, otherwise it could wait forever for release and fail to achieve its goals. On the other hand, should A_2 have been expected to *release* A_1 in the prior solution, but now that requirement is lifted, the disruption to A_2 would be less, because sending a *release* that is no longer being *wait*ed for only wastes time and bandwidth, rather than leading to deadlock. We thus assign different values, δ_w and δ_r, for the dropped *wait* and *release* modifications respectively. As per the above, we would qualitatively expect that $\delta_w > \delta_r$.

When a commitment in the new solution has no corresponding commitment in the old, we need to include the disruption caused by the added unmatched commitment. Since we have seen that dropping *release* and *wait* commitments might cause different amounts of disruption, it may also be that adding different synchronization commitments might have different disruption. We assign the value δ_x to an added *wait* commitment's disruption, and δ_s for an added *release* commitment. We would expect that the disruption of adding a new synchronization commitment would be at least as great as a totally mismatched commitment, and so expect that its value (δ_x or δ_s) will be at least $\delta_e+\delta_c+\delta_a+\delta_o$.

3.1.2 Multiple Synchronization Commitments

It is likely that there are multiple commitments in either the old or new solution (or both). In this case, we should match the plan commitments that are most similar to each other to get a true sense of the disruption. Clearly, we want to match identical commitments to each other, since they would not add any disruption to the agent. By matching the remaining pairs in order of minimized distance between them, we have a consistent method for pairing the commitments. When the number of plan commitments is not the same, there will be some left unmatched, and these will contribute to disruption as just described.

We use a greedy algorithm (Figure 3) that calculates the disruption (Ds_i) to an agent (A_i) caused by switching from one set of commitments (S_i') to another (S_i). It first finds identical matches between the sets of commitments (checks for disruption=0), and then, among the unmatched commitments, successively looks for the best remaining match between the sets. Each pair of matched commitments (S_{ij} and S_{ik}') will add the appropriate amount (Ds_{ijk} where k is actually dependant on j) to the disruption score (Eq. 5).

```
// For each agent A(i) examine S(i) and S'(i)
Set "check value"=0
While elements remain in both sets {
 For each element in set S {
  For each element in set S' {
   Calculate disruption between elements
   If disruption="check value" {
    Add disruption to total
    Remove both elements from sets
    Continue with next element in S }}}
 Increment "check value" }

// One or both sets are now empty
For each element remaining in S or S' {
 Add appropriate unmatched disruption value }
```

Figure 3: Calculate Disruption for Sets of Commitments

The remaining unmatched commitment(s) from the new or old solution list will add their respective values (δ_x, δ_s, δ_w *or* δ_r).

Sorting the lists of commitments, we can present the following equations that summarize the calculations done in the algorithm, where n is the number of new commitments, and o is the number of old commitments.

When o>n:

$$Ds_i = \sum_{j=1..n} Ds_{ijk(j)} + \sum_{k=n+1..o} (\delta_w(S_{ik}'[2]==\text{"wait"}) + \delta_r(S_{ik}'[2]==\text{"release"})) \quad (6)$$

When n>o:

$$Ds_i = \sum_{j=1..o} Ds_{ijk(j)} + \sum_{j=o+1..n} (\delta_x(S_{ij}[2]==\text{"wait"}) + \delta_s(S_{ij}[2]==\text{"release"})) \quad (7)$$

The first sum over j in each case is for the matched commitments, and the second sums are intended for the unmatched commitments. In the first sum, Ds_{ijk} uses S_{ik}', the appropriately matched old commitment for S_{ij}, so k depends on j.

Note that if $S_i=S_i'$, then $Ds_i=0$. Note also that, though calculating the disruption of a single pair of synchronization commitments is reflexive, the disruption of two sets of commitments may not be, since the two sets may be of different sizes, and the δ values for different kinds of unmatched commitments may not be equal.

3.2 Blocking Commitments

Since blocking commitments are distinctly different from synchronization commitments, they are treated separately. A blocking commitment is represented by a 3-tuple (*agent*, *plan-step*, *parent-plan-step*). Since a blocking commitment only deals with a single agent, old and new commitments either refer to identical plan steps, or else they do not. When they do, they match. If they do not, then the disruption should also examine the ancestry of the plan steps. For example, if the old and new plan steps share a common parent (that is, they are alternative *or* branches of the same abstract plan step), then that information can be used to estimate the disruption more accurately.

Returning to our simplified example (Figures 1 and 2), consider the following progression of commitments. Initially, A_1 might be alone in the world, and thus would adopt its own plan (Figure 2), leaving itself flexibility as to whether to use the route through B or the route through C. If A_2 enters the world and has the identical plan, now there is the possibility of collision along one of the routes. This can be averted by having agents make blocking commitments, such as A_1 committing to not move

through C, and A_2 committing to not move through B. From the perspective of A_1, this disrupts its prior expectations of flexibility; however, it could be that A_1 had already decided to move through B anyway, so the disruption is minimal. Either way, we would want to estimate this disruption.

Further, let us say that A_2, for some reason, changes its mind and determines that it really needs to move via B. Now A_1 is perhaps more strongly disrupted in its expectations, needing to definitely adopt the opposite plan than what it had planned to do. We would expect that this would disrupt A_1 more severely, therefore. Finally, though, let us say that A_2 decides to abandon its goal entirely, leaving the world entirely to A_1. A_1 can thus remove its commitment to block the route via B. Of course, it could simply pretend like the commitment was still there, and continue to pursue a plan to move via C, so it need not be strongly disrupted.

Thus, the degree to which changes to blocking commitments will disrupt agents depends on whether these commitments are being added, removed, or switched. To capture these differences, we use δ_b to model the costs of adding a blocking commitment, and δ_u for a removed commitment, although from our simple analysis above it seems reasonable to assume that $\delta_u=0$. If a blocking commitment is removed while an unrelated commitment is added, we would expect the disruption to equal the sum of the disruptions of the two changes to commitments separately. However, if the commitments being added and removed are related, such as switching the blocking of an action with blocking its sibling, then the disruption could be higher than the sum of adding and removing constraints because an agent is forced away from an action it could have planned toward an action that it would not have planned. We use δ_m for the disruption of a switched blocking commitment between sibling plan steps, and δ_n for a non-sibling change to blocking commitments.

In general, the disruption of a new commitment relative to a prior commitment can be directly calculated from the two commitments by comparing the plan step attributes:

$$Db_{ijk} = \delta_m((B_{ij}[1] \neq B_{ik}'[1]) \,\&\, (B_{ij}[2] = B_{ik}'[2])) + \delta_n((B_{ij}[1] \neq B_{ik}'[1]) \,\&\, (B_{ij}[2] \neq B_{ik}'[2])) \quad (8)$$

where $B_{ij}[0]$, the first attribute, is the agent and must be the same for the two commitments. $B_{ij}[1]$ is the plan step being blocked, and $B_{ij}[2]$ is the parent of the plan step $B_{ij}[1]$, which allows us to identify sibling plan steps.

3.2.1 Blocking Commitment Representation

Although blocking commitments, like synchronization commitments, are imposed to successfully coordinate, they differ in that they do not refer to the other agents with whom an interaction is foreseen (in contrast with synchronization commitments, where the agent waiting and the one releasing are explicitly represented). The reason for this difference is fundamental. A blocking commitment is an indication that a child of an *or* node is removed from the final coordinated plan because a conflict was identified between the conditions of the *or* node and the conditions of another agent's plan step. However, there may be more than one agent impacted from the blocking commitment. For example, if A_1 has blocked the route via C (Figure 1) to avoid collision with A_2, then agents that join later can also take advantage of this commitment when planning their routes. Later, even if A_2 changes its plans entirely, A_1 might still need to adhere to the blocking commitment for the sake of other agents. If the commitment explicitly modeled for whom the blocking was being done, then this kind of change would require a change in commitments being modeled. However, since, either way, A_1 is blocking the same plan steps, there should be no disruption from such a change. Hence, this information is not included in the commitment representation.

3.2.2 Multiple Blocking Commitments

Because coordination might require multiple blocking commitments, we need to match them into pairs (to the extent possible) to compute disruption. Each agent (A_i) is subject to a set of (b) blocking commitments $B_i = \{B_{i1}, B_{i2}, \ldots, B_{ib}\}$. Given two sets of blocking commitments (B_i and B_i'), we can calculate the disruption (Db_i) to an agent (A_i) that switching from one set of commitments (B_i') to the other (B_i) would cause. The algorithm used is the same as that for synchronization commitments, although it operates on the B sets instead of the S sets and a different evaluation function is used. Similarly to the Ds_i equations (Eq. 6 & 7) we present the following Db_i equations, where n is the number of new commitments, and o is the number of old commitments.

When $n>o$:

$$Db_i = \sum_{j=1..o} Db_{ijk(j)} + \sum_{j=o+1..n} \delta_b \quad (9)$$

When $o>n$:

$$Db_i = \sum_{j=1..n} Db_{ijk(j)} + \sum_{k=n+1..o} \delta_u \quad (10)$$

Note that calculating the disruption of two sets of blocking commitments is not reflexive, since the two sets may be of different sizes and δ_b need not equal δ_u.

3.3 Estimating Overall Disruption

For any agent A_i in the set of agents $A = \{A_1, A_2, \ldots, A_n\}$, the associated sets of synchronization commitments are $S_i = \{S_{i1}, S_{i2}, \ldots, S_{is}\}$ and blocking commitments are $B_i = \{B_{i1}, B_{i2}, \ldots, B_{ib}\}$. S_i and B_i could each or both be empty. The disruption to an agent (D_i) is a function of the new commitments (S_i and B_i) for the agent when compared to the old commitments (S_i' and B_i') for the same agent. The total disruption to a single agent is simply:

$$D_i = Ds_i(S_i, S_i') + Db_i(B_i, B_i') \quad (11)$$

Where Ds_i and Db_i have been defined earlier (Eq. 6, 7, 9 and 10) and can be weighted relative to each other by weighting the deltas for synchronization commitments (δ_e δ_c δ_a δ_o δ_w δ_r δ_x δ_s) relative to those for blocking commitments (δ_u δ_m δ_n δ_b).

The total disruption (D) of the new coordination solution (C) relative to the old (C') is just the sum of the disruptions to all the agents involved in the new solution.

$$D(C,C') = \sum_{i=1..n} Ds_i(S_i, S_i') + Db_i(B_i, B_i') \quad (12)$$

The disruption calculation should guarantee that the better the commitments match, the lower the disruption will be. However, while we have defined constraints (inequalities) on some of these parameters, we have not suggested numeric values for them. Ultimately, these parameter values summarize an expected disruption cost that will be extremely domain dependent. While we will use some numeric values for our experiments later, our formulation purposely leaves these as user tunable parameters.

4. SORTING CANDIDATE SOLUTIONS

As solutions are generated by the top-down coordination process [1], they will each have an associated disruption value. We create a list of the solutions as they are generated and sort according to the disruption values. Whoever selects the adopted solution then has the ability to easily see the top candidates. We do not assume that disruption will be the only factor involved in selecting a solution, but it could be important.

To verify that our implementation is working properly, we can run the system with the same sets of agents and plans as were used to select the prior (reference) solution. As expected, the same solution always comes out with zero disruption. The only other solutions that have zero disruption are trivially identical solutions (one or more *and* nodes, which have no other constraints on them, are refined). The solutions with low (but non-zero) disruption are also very similar to the reference, and predominantly solutions where an *and* node with a constraint on it has been refined (so that the constraint now applies to a child plan step of the node).

Disruption can be used as a measure of quality of the solution, but we do not suggest that it be the only measure of quality used. Other measures of quality could be the length of time the overall solution requires at execution time (the level of concurrency), and the flexibility of the plans (retaining its *or* branches). We expect that an expert would select a solution that simultaneously minimizes the overall execution time, maximizes the plan flexibility, and minimizes the disruption to the agents. The first two can generally not be satisfied simultaneously, but the choice for the prior solution is typically preserved in the least disruptive subsequent solution. That is, when we start with a reference solution that has the shortest execution time, the solutions with the least disruption are among the shortest duration. Likewise, starting with a reference solution that is more flexible (generally more abstract), the solutions with the least disruption are among the most flexible. Qualitatively, disruption values make sense.

Although the implementation works as expected, doing the top-down hierarchical search for solutions is no faster in finding the least-disruptive solution (which from this point on in this paper we refer to as the “best” solution) than it was the first time around. If the best solution is a more abstract (serialized) solution, it is generated near the start of the search. If the best solution involves constraints between low-level primitives to achieve concurrency, it will likely be generated late in the search. If low-disruption solutions are more desirable, and solutions might be selected under time pressure, we would like to generate them earlier in the search - we need to bias the search process towards finding them sooner.

5. IMPROVING SOLUTION GENERATION

Because a coordination solution consists of a set of partially-refined, single-agent hierarchical plans along with a set of coordination commitments, the coordination process must be capable of taking, as input, the plan hierarchies of the agents, and searching for these solutions. The coordination search involves two nested phases, an outer search though plan refinements, and an inner search for inter-agent coordination commitments [1,2]. The outer search iteratively refines abstract plan steps to generate the partially-refined hierarchies. Each of the refinements can be thought of as a cut through (or frontier of) the plans' hierarchies. When an *or* node is refined, furthermore, there will be multiple successor refinements, each representing the selection of one of the possible refinement plan steps, and thus committing to blocking the other possible refinements.[2]

Given any particular combination of plan refinements, there may still be multiple ways to impose synchronization commitments between the steps to produce a coordinated solution, so a second inner search is performed on each frontier to find these. This search compares plan steps of different agents in all possible combinations to detect possible conflicts, and generates all ordering commitment possibilities that would avoid the detected conflicts. Note that, if the agents' plans are flat (not hierarchical), then there is no outer search process, and the method described here does not speed up the search for good solutions, but is still able to rank solutions according to the amount of disruption.

By default, the search for coordination solutions works top-down through the agents' plan hierarchies, attempting to find ways of resolving conflicts at more abstract levels (which involve fewer, larger plan steps) before seeking resolutions at more detailed levels (which can require substantially more effort to find). It works by initializing the outer search with the frontier containing all agents' most abstract plan steps, passing this into the inner search to generate all feasible solutions at the abstract level, then returning to the outer search to generate the next refinement of the frontier, passing that result to the inner search, and so on. This process generates solutions that are then sorted; compared to the solution generation process, the ranking and sorting process only adds an additional 3% of time and memory.

However, if the goal is to find less disruptive solutions more quickly, it should be possible to work not "top-down" but rather to radiate out from more likely candidates. Specifically, because coordination commitments involve the selection of particular refinements (for blocking undesirable ones) and imposition of synchronization commitments between particular plan steps, then minimally disrupting solutions should involve the same plan steps as the reference solution, to the extent possible. Refinements that have not gone far enough, or have gone too far, or have refined along very different *or* branches, hold less promise of yielding less disruptive solutions. The search should postpone the expensive inner search process on those frontiers until after more promising candidates are tried.

Rather than automatically passing the outer search nodes into the inner search as they are generated, therefore, the outer search nodes are preliminarily inspected for disruption. Because an outer search specifies a particular frontier of refined plan steps, and because disruption due to blocking only needs this information for each agent, part of the overall disruption measure can be found for the outer search node. A lower bound on the disruption due to synchronization commitments can also be estimated and added in. Numerous outer search nodes can thus be generated and ranked before any inner search is done, and thus earlier applications of the inner search are more likely to give less disruptive solutions.

---

[2] For simplicity, we are not modeling more complex blocking constraints. When there are more than two children, blocking constraints considering all possible subsets of children of an *or* node could result in additional solutions, but those solutions could be reconstructed from solutions already being generated.

Furthermore, if minimal disruption is the only solution criterion, then branch-and-bound can be used to prune the search. A node generated by the outer search whose disruption solely based on blocking equals or exceeds the least disruptive solution found so far need never itself undergo the inner search. However, a child of that node in the outer search space might have a lower value of disruption (if the outer search node had not been refined far enough), so a node that will not undergo inner search should still have its successor (refined) nodes generated and evaluated. In the future, we will study how we can entirely prune nodes even in the outer search by knowing when none of their successors could possibly have lower disruption. This has not been of primary importance since the outer search is much less time-consuming than the (combinatorial) inner search.

5.1 Implementation

The new methods for measuring disruption, and biasing/pruning search based on minimizing disruption, have been implemented in the Multilevel Coordination Agent (MCA) [1,3]. The general outer search in the original code was functionally similar to that reported here, but had hardwired the top-down search and so required some rewriting to accommodate heuristic ordering procedures and pruning mechanisms. The actual functional differences that resulted are only those needed to bias the search towards nodes in the outer search space that have promisingly low levels of disruption, while also allowing the pruning of nodes with high disruption.

```
Initialize best score (infinity)
Calculate score and priority for the root node
Place the root node on the queue.
Place the root node on the closed list
While there are nodes on the queue {
 Pull a node from the priority queue.
 Generate children of node
 For each child {
  If the child is not on the closed list {
   Calculate score for child
   If child score <= best {
    Place child node on queue
    Place the child on the closed list }}}
 If the node's score <= best {
  Perform inner search (output solutions)
  Determine best score from new solutions }}
```

Figure 4: Biased Search and Pruning Algorithm

The general search process that is performed is given as pseudo-code (Figure 4). The high-level functions referenced there are described as follows.

Calculate Score: Calculates the disruption for a node as compared to a reference set of commitments. When calculating the score for a solution, both the disruption caused by changes to the blocking commitments for each agent, and the disruption caused by changes in the inter-agent synchronization commitments, are included. When calculating the score of an outer search node, the actual blocking commitments are considered, along with an estimate for synchronization commitments.

The inner search adds inter-agent synchronization commitments between plan steps and agents. These commitments (for the reference solution) are compared to the plan steps and agents that exist in the current node. The best possible match is used for each commitment to give a lower bound for the disruption that could be generated in the inner search.

Place node on queue: The estimated disruption for an outer search node is used to insert the node into a priority queue. The node score is thus used to bias the search towards refinements that will allow the inner search to find a low disruption solution.

Perform inner search: A search is performed with the current node to find inter-agent ordering commitments that resolve any conflicts remaining between the plans in the node. The solutions will consist of the agents' original plans and a set of commitments for those plans that detail the inter-agent synchronizations added in the inner search, and the blocking commitments (if any) added in the outer search. Each of the solutions will have a disruption score associated with it.

6. SEARCH SPEED IMPROVEMENT

A main challenge we face in recoordination is dealing with the entry of an unexpected agent whose plan conflicts with the already coordinated solution. While it is clear that all sorts of other changes to a system can be disruptive, adding an agent (or multiple agents) is the situation where the computational effort goes up most dramatically in recoordination, and reducing that increase becomes important.

The agent plans we use in this evaluation are identical to our original plan (Figure 2), but are generated automatically with random names for the intermediate points. This models a number of agents all competing for flight corridors in a limited air space. Each agent accomplishes an abstract goal in one of two ways, requiring two distinct resources (the flight corridors) in each case. Other agents will have identically structured plans, where there may be conflicts over the flight corridors. We can vary three parameters in generating our test plans: the number of agents, the number of plan steps conflicting with other agents, and the number of agents that the conflicting plan steps conflict with.

Although we presented extreme levels of conflict between the agents in our initial example (all plan steps conflicting with all other agents), this is unrealistic. Instead, what is more likely to occur is that only a small portion of an agent's plan will conflict with the other agents' plans. To model this more realistic level of overlap between the plans, our first test generates plans where only one plan step for an agent conflicts, but with all other agents. In this first case (the two solid curves in Figure 5), we look at how varying the number of agents affects the overall time required to coordinate the plans using biased vs. unbiased search. The time to find all solutions in the unbiased search increases exponentially with the number of agents. Biased search prunes parts of the search space and, although it also increases exponentially, the slope on the log scale is slightly less, 1.2 instead of 1.5. When searching for the least disruptive solution in unbiased search, there is no way to be sure it is the least disruptive without an exhaustive search. With a biased search, the least-disruption solution is verified faster because the number of possible solutions generated, and the total search time, are reduced.

In all cases, the least disruptive solution was found approximately halfway through the entire unbiased search process, while it was found one-fifth of the way through the biased search process. Because the search must complete before we can guarantee that the least disruptive solution has been found, and because the speedup is only linear, detailed results are not presented here.

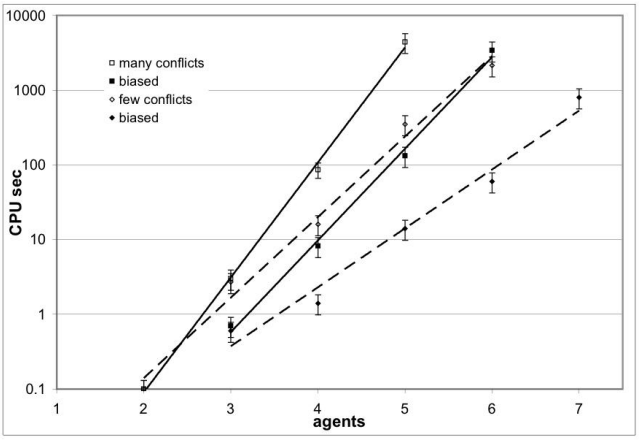


Figure 5: Comparison of times required to find solutions with and without disruption based bias and pruning.

In the second case (the two dashed curves in Figure 5), the agents are subjected to fewer conflicts; the conflicting plan step only conflicts with one or two other agents. The points do not appear to follow as straight a line because of even/odd numbers of agents. With an even number of agents, we can limit the conflicts between agents to a single other agent. With an odd number, the extra agent must conflict with two other agents, also increasing their conflicts, and making the coordination problem more difficult. Again, the biased search is an order of magnitude faster than the unbiased search, and the slope is 0.8 instead of 1.0. In general, we can prune away enough nodes using bias to finish the search for solutions for *N* agents in much the same time as the unbiased search does for *N-1* agents. With larger numbers of agents, the relative timesavings gets even greater.

7. Conclusions And Future Work

In this paper, we described a method for measuring disruption to agents facing recoordination. We have shown how this flexible measure can be used to bias and bound the search process to find suitable coordination solutions faster.

Although our research focused mainly on how using disruption measures can speed up the recoordination problem, we have seen that there are instances where it could be used to speed up the initial coordination problem. For instance, coordinating *N-1* agents and then recoordinating *N* with bias can be an order of magnitude faster than coordinating the *N* agents without bias (when $N>3$). We are interested in applying disruption measures to a wider range of problems, and intend to evaluate using disruption to coordinate larger numbers of agents by recoordinating groups of locally coordinated agents.

Although our work so far has only included synchronization and blocking commitments in measuring disruption, we intend to add reasoning about temporal constraints to the disruption measure. When agents have synchronized their plans against clock times, changing those time constraints could also be disruptive to the agents and their other commitments. When appropriate, the added measures of disruption could make the biased search even faster.

Our current work on disruption is closely tied to HTN planning, and the kinds of commitments used in our metrics may not be applicable to all planning methods. In a wider context, applying the techniques described here to other plan coordination mechanisms should be possible.

8. Acknowledgments

This work benefited from discussions with Jeff Cox and Brad Clement and the code that they have written. DARPA (F30602-98-2-0142) supported this research.

9. References

[1] B. Clement and E. Durfee. "Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents." *Proceedings of the Third International Conference on Autonomous Agents*, 252-259, 1999.

[2] B. Clement, "Abstract Reasoning for Multiagent Coordination and Planning" PhD thesis, University of Michigan, 2001.

[3] J. Cox, B. Clement, P. Pappachan, and E. Durfee. “Integrating Multiagent Coordination with Reactive Plan Execution.” *Proceedings of the Fifth International Conference on Autonomous Agents* , 149-150, 2001.

[4] E. Durfee, and V. Lesser, "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation," *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167-1183, September/October 1991.

[5] N.R.Jennings, "Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems", *The Knowledge Review*, 8(3), pp 223-250, 1993.

[6] D. Kinny and M. Georgeff. "Commitment and effectiveness of situated agents." *Proc. of IJCAI-91*, pp. 82-88, 1991.

[7] M. Pollack and J. Horty, "There's More to Life than Making Plans," *AI Magazine*, 20(4):71-84, 1999.

[8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, pp 392-410, Prentice Hall, New Jersey, 1995.

[9] E. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier/North-Holland, Amsterdam, London, New York, 1977

[10] A. Tate, "Generating Project Networks," *Proceedings of the Tenth Intl. Joint Conference on Artificial Intelligence*, 1987.

[11] T. Tsukada, and K. Shin, "PRIAM: Polite Rescheduler for Intelligent Automated Manufacturing" in *IEEE Trans. Robot. Automat*., vol.12, pp. 235-251, April 1996.

[12] G. Weiss, ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge Massachusetts, 1999.

[13] Control of Agent Based Systems Project http://coabs.globalinfotek.com

[14] Vipin Kumar, "Algorithms for Constraints Satisfaction problems: A Survey", *AI Magazine*, 13(1):32-44, 1992.

Appendix I:

Coalition Agents Experiment: Multi-Agent Co-operation in an International Coalition Setting

David N. Allsopp[1], Patrick Beautement[1], Jeffrey M. Bradshaw[2], Edmund H. Durfee[3], Michael Kirton[1], Craig A. Knoblock[4], Niranjan Suri[2], Austin Tate[5], Craig W. Thompson[6]

1. QinetiQ Ltd,
Malvern Technology Centre,
St Andrews Road, Malvern,
Worcestershire, WR14 3PS, UK
{d.allsopp, m.kirton}@signal.qinetiq.com, pbeautement@qinetiq.com

2. Institute for Human & Machine Cognition
University of West Florida
40 South Alcaniz Street
Pensacola, FL 32501, USA
{jbradshaw, nsuri}@ai.uwf.edu

3. EECS Department
University of Michigan
Ann Arbor, MI 48109, USA
durfee@umich.edu

4. Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
knoblock@isi.edu

5. Artificial Intelligence Applications Institute
Centre for Intelligent Systems and their Applications
Division of Informatics, The University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK
a.tate@ed.ac.uk

6. Object Services and Consulting, Inc. (OBJS)
2725 Deep Valley Trail, Plano, TX 75023, USA
thompson@objs.com

Abstract. Military Coalitions are examples of large-scale multi-faceted virtual organizations, which sometimes need to be rapidly created and flexibly changed as circumstances alter. The Coalition Agents eXperiment (CoAX) aims to show that multi-agent systems are an effective way of dealing with the complexity of real-world problems, such as agile and robust Coalition operations and enabling interoperability between heterogeneous components including legacy and actual military systems. CoAX is an international collaboration carried out under the auspices of DARPA's Control of Agent-Based Systems (CoABS) program. Building on the CoABS Grid framework, the CoAX agent infrastructure groups agents into domains that reflect real-world organizational, functional and national boundaries, such that security and access to agents and information can be governed by policies at multiple levels. A series of staged demonstrations of increased complexity are being conducted in a realistic peace-enforcement scenario situated in 2012 in the fictitious African state of "Binni". These demonstrations show how agent technologies support the rapid, co-ordinated construction of a Coalition command system for intelligence gathering, for visualization, and for campaign, battle and mission planning and execution.

1 Introduction and Background

1.1 Military Context

Success in military operations involves carrying out high-tempo, coherent, decisive actions faster than an opponent can react, resulting in decision dominance through the use of command agility. Command agility is about being flexible and adaptable so that fleeting opportunities can be grasped; the Commander issues clear intent and then delegates control to subordinates, allowing them the scope to exercise initiative. It also means being innovative, creative and unpredictable in a manner that (even if low-tempo) increases confusion in the mind of an opponent. This process is command led; human decision-making is primary and the role of technology is secondary. Shared understanding and Information Superiority are key enablers in this process and are fundamental to initiatives such as

the UK's Command and Battlespace Management program, the US Joint BattleSpace Infosphere program and, more generally, Network-Centric Warfare (http://www.dodccrp.org/).

In addition to the problems of integrating single-service and Joint capabilities into a coherent force, the nature of Coalition (multi-national) operations implies some need to rapidly configure foreign or 'come-as-you-are' systems into a cohesive whole. Many problems in this environment can only be solved by organizational changes and by 'aligning' doctrine, concepts of operations and procedures. Due to the inevitable absence of pre-existing co-ordinated systems, Coalition scenarios require a rapid, flexible, on-the-fly approach that allows capabilities to be assembled at run-time. However, in addressing this requirement for interoperability, it is also crucial to address issues of security of data, control over semi-trusted software from other Coalition partners, and robustness of the resulting system (e.g. the ability to withstand denial-of-service attacks).

Currently, Coalition operations are often characterized by data overload, information starvation, labor intensive collection and co-ordination of information, and standalone stove-pipe command systems that use incompatible data formats. This leads to a horrendous technical integration task and gives commanders only scattered snapshots of the battlespace. This paper aims to show that the agent-based computing paradigm offers a promising new approach to dealing with such issues by embracing the open, heterogeneous, diverse and dispersed nature of the Coalition environment. In this paper, we show that software agents that act on behalf of human users enable military commanders to act decisively in cyberspace and thus contribute towards the achievement of 'Cyberspace Superiority', a critical component of warfare in the information age (Alberts et al, 2001).

1.2 Software Agent Technology

Software agents are currently receiving much attention in the research community. This interest is being driven by the phenomenal growth of the Internet and the World-Wide-Web. Agents can be viewed as semi-autonomous software designed to help people cope with the complexities of working collaboratively in a distributed information environment. This involves the agents communicating between the users and between themselves. The agents are used to find, format, filter and share information, and work with users to make the information available wherever and whenever they need it. The agents are also able to proactively suggest courses of action, monitor mission progress, and recommend plan adjustments as circumstances unfold.

A community of agents can be seen as a set of distributed, asynchronous processes communicating and sharing information by message passing in some infrastructure. In this regard, an important output from DARPA's CoABS program is the CoABS Grid — a middleware layer based on Java / Jini technology that provides the computing infrastructure to integrate heterogeneous agent communities and systems rapidly (http://coabs.globalinfotek.com/).

A recent article (Jennings, 2001) argues that the agent paradigm is a good way of building complex software systems in general, and hence offers potential benefits in the Coalition setting. For example, legacy command systems could be provided with software agent wrappers that allow them to inter-operate and share information with other systems and agent applications in a loosely connected, heterogeneous architecture, underpinned by the CoABS Grid. The scenario used as the basis of the experiments to test this hypothesis is described in section 2.

1.3 Aims of the CoAX Project

This paper describes the progress of an international collaborative effort whose overall goals are to demonstrate that the agent-based computing paradigm offers a promising new approach to dealing with the technical issues of establishing coherent command and control (C2) in a Coalition organization. This collaborative effort, entitled CoAX (Coalition Agents eXperiment), is a Technology Integration Experiment under the auspices of DARPA's Control of Agent Based Systems (CoABS) program (http://www.aiai.ed.ac.uk/project/coax/). Specific hypotheses of the research program are that:

- agents are a useful metaphor for dealing with the complexity of real-world systems such as military operations;
- an agent-based C2 framework can support agile and robust Coalition operations;
- software agents can be used to enable interoperability between legacy or previously incompatible systems;
- the CoABS Grid can be used to rapidly integrate a wide variety of agents and systems — i.e., rapid creation of virtual organizations;
- domain policies can structure agent relationships and enforce Coalition policies;
- intelligent task and process management can improve agent collaboration;
- semantic web technology can improve agent interoperability between disparate Coalition command systems.

The CoAX team has built a software agent test-bed based on the CoABS Grid (http://coabs.globalinfotek.com/). This paper describes the work done, the demonstrations carried out so far, the scenario and storyboard used and some of the insights gained.

1.4 Structure of the Paper

The paper begins by describing the Coalition scenario and military command structure used in our demonstration experiments. Section 3 describes the corresponding agent architecture that was developed to reflect the military organizational structure. The events occurring in the storyboard used for the various demonstrations so far are described in Section 4. A preliminary assessment of software agent capabilities and a discussion of future research are provided in Section 5. Concluding remarks are given in Section 6.

2 A Representative Scenario and Coalition Command Structure

The CoAX work needed a suitably realistic scenario for its experiments and so we expanded the fictional "Binni" scenario (Rathmell, 1999) developed for The Technology Co-operation Programme. In this scenario the year is 2012 and global warming has altered the political balance of the world. The action is set in an area that is currently the Sudanese Plain (Figure 1). Previously uninhabited land in the Plain is now arable and the area has received large amounts of foreign investment. It is now called “The Golden Bowl of Africa”.

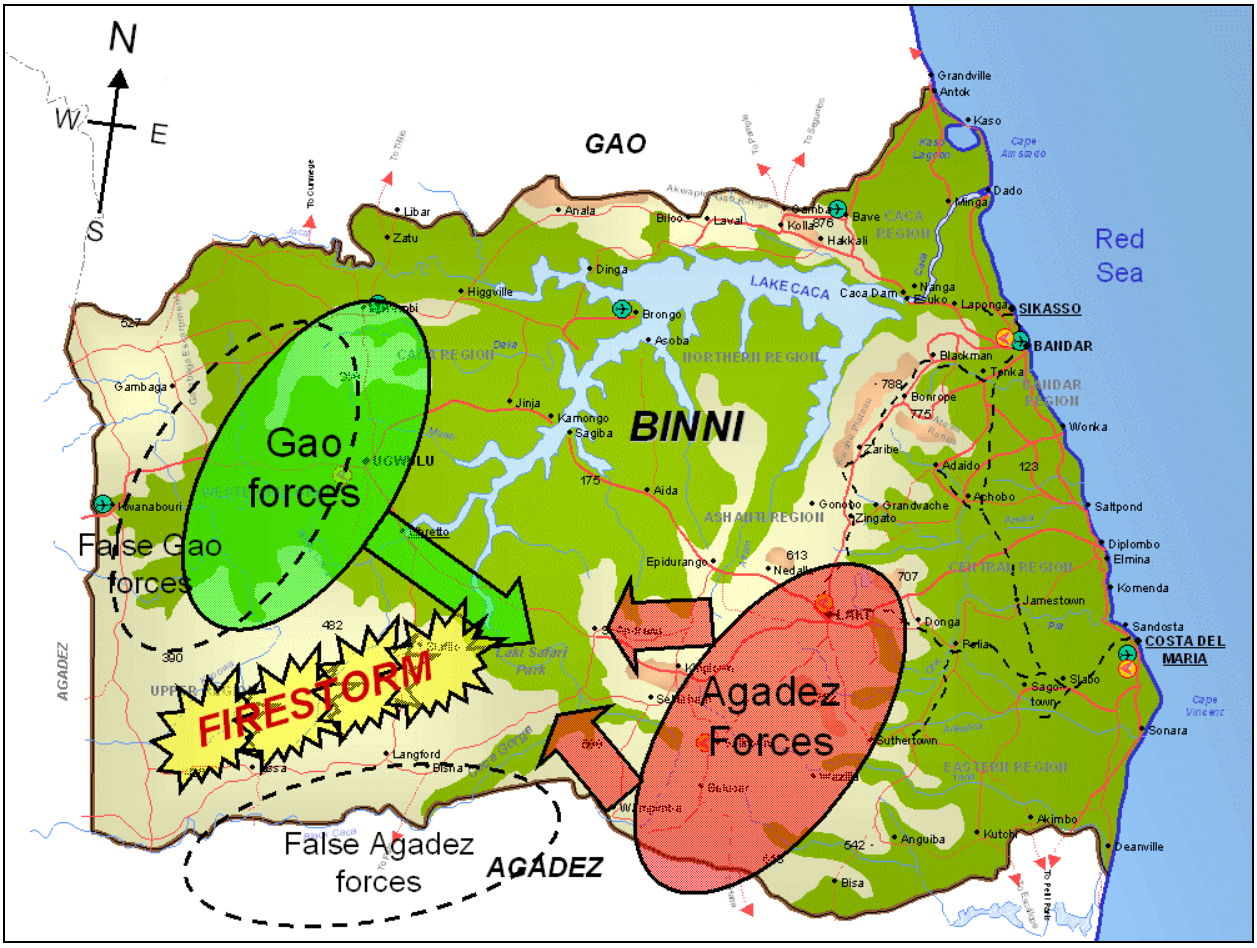


Figure 1. Map of Binni showing firestorm deception. Misinformation from Gao is intended to displace the firestorm to the west, allowing Gao and Agadez forces to clash in the region of the Laki Safari Park.

A conflict has developed between two countries in the area. To the north is Gao, which has expansionist aspirations but which is only moderately developed, with old equipment and with a mostly agrarian society. To the south is Agadez, a relatively well developed and fundamentalist country. Gao has managed to annex an area of land, called it Binni and has put in its own puppet government. This action has come under fierce attack from Agadez. Gao has played the ‘threat of weapons of mass destruction from Agadez’ card and has enlisted support from the UN who have deployed a force, the UN War Avoidance Force for Binni (UNWAFB), to stabilize the region. This basic scenario was adapted for a number of CoAX demonstrations (see Section 4), beginning with the initial planning phase, then moving onto shorter timescales and more dynamic, uncertain events for the execution phase.

2.1 Coalition Command Structure

This Binni Coalition operation needs to rapidly configure various incompatible, ‘come-as-you-are’ or foreign systems into a cohesive whole within an open, heterogeneous and dispersed environment. This scenario provides a

suitable test for the software agent experiments, where run-time composability is a very close metaphor for the dynamic uncertainty of Coalition operations. The complexity of the situation must not be underestimated and is best illustrated by looking at the Binni Coalition Command Structure shown in Figure 2 below.

This is a representative and realistic Coalition command structure involving the UN, Governments, Other Government Departments (OGDs, such as the Foreign Office), Non-Government Organizations (NGOs, such as Oxfam), representatives of all the Coalition countries (with their own 'ghosted' Command Structures) and the Coalition HQs and subordinate fighting forces. The solid black lines on the diagram show the legal lines of authority (the command chain) and accountability. This is the kind of Coalition structure that would be agreed by the participants; no part of the formal command chain is owned by any specific country. Note that the 'levels of command' overlap and their boundaries are not rigidly defined. Dashed lines show an advisory / negotiating role.

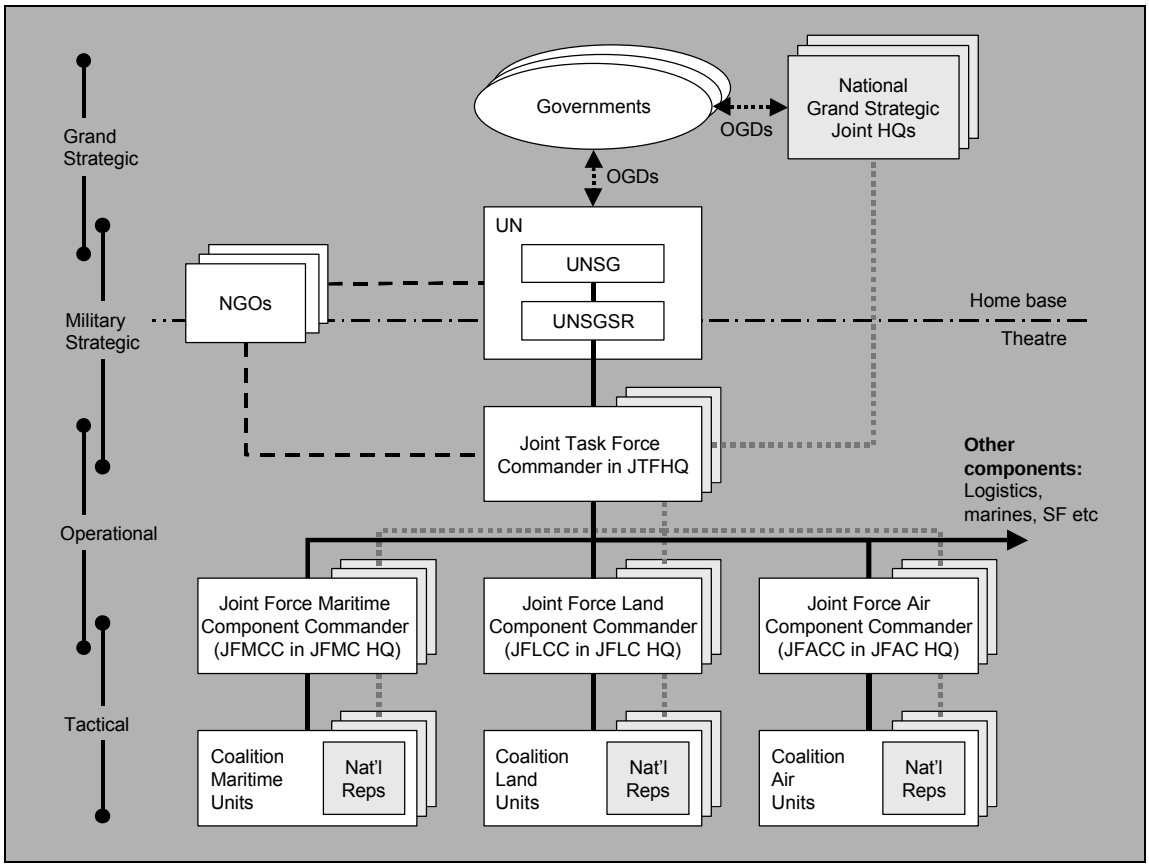


Figure 2: A representative Coalition structure, showing the chain of command down from the United Nations, including the 'ghosted' command structures of the participant nations, and Non-Government Organizations (NGOs). The approximate command level at which the various entities operate is indicated on the left.

3 Software Agent Architecture

3.1 Human Domains

Integrating information across a Coalition is not just a matter of employing technology — it involves the creation of a coherent 'interoperability of the mind' at the human level as well, where many social and cultural factors come into play. The mapping between the human and technical worlds is thus not straightforward. From the human perspective, we identified four kinds of 'domains':

- **Organizational Domains**: for example the Joint Task Force HQ (JTF HQ) ;
- **Country Domains**: each of the National command chains would be a separate, self-contained domain;
- **Functional Domains**: sets of entities collaborating on common tasks, for example Meteorology or Intelligence ;
- **Individual Human Domains of Responsibility**: Commanders have responsibility for their own HQ and all subordinate ones (in practice they delegate). Hence the individual human domains of influence may overlap.

These types of domains are not entirely exclusive and there are many different levels of overlap and interaction depending on the viewpoint taken. It is this complexity at the human level that creates difficulties for technical systems.

3.2 Software Agent Domains

3.2.1 CoABS Grid Infrastructure

At the most basic level, the agents and systems to be integrated require infrastructure for discovery of other agents, and messaging between agents. The CoABS Grid provides this. Based on Sun's "Jini" services which are themselves based upon Java's Remote Method Invocation, the Grid allows registration and advertisement of agent capabilities, and communication by message-passing. Agents on the Grid can be added or removed, or their advertisements updated, without reconfiguration of the network. Agents are automatically purged from the registry after a short time if they fail. Multiple lookup services may be used, located by multicast or unicast protocols. In addition, the Grid provides functionality such as logging, visualization, and more recently encryption of messages and agent authentication.

3.2.2 KAoS Domain Management

The increased intelligence afforded by software agents is both a boon and a danger. By their ability to operate independently without constant human supervision, agents can perform tasks that would be impractical or impossible using traditional software applications. On the other hand, this additional autonomy, if unchecked, also has the potential of effecting severe damage to military operations in the case of buggy or malicious agents. The Knowledgeable Agent-oriented System (KAoS) provides services that help assure that agents from different developers and running on diverse platforms will always operate within the bounds of established policies and will be continually responsive to human control so that they can be safely deployed in operational settings (Bradshaw et al., 1997, 2001). KAoS services and tools are intended to allow for the specification, management, conflict resolution, and enforcement of policies within the specific contexts established by complex military organizational structures.

KAoS domain management services can be used to group agents into logical domains corresponding to organizational structures, administrative groups, and task-oriented teams. Within CoAX, these domains mirror the human domains described above, allowing for complex hierarchical, heterarchical, and overlapping structures. An agent domain consists of a unique instance of a domain manager (DM) along with any agents that are registered to it. Alternatively, an intensionally-defined domain consists of a set of agents sharing one or more common properties (e.g., the domain of all agents physically residing on some host). The function of a domain manager is to manage agent registration, and serve as a single point of administration and enforcement for domain-wide, host-wide, VM-wide, VM-container-wide, or agent-specific policies.

3.2.3 Domain policies

A policy is a declarative constraint governing the behavior of one or more agents, even when those agents may not be domain-aware or where they may be buggy or malicious. For example, a policy may be declared that all messages exchanged among agents in the JFAC HQ domain must be encrypted, or that an agent cannot simultaneous belong to the US and the UK domain. A policy does not tell the agent how to perform its task; it rather specifies the conditions under which certain actions can be performed. By way of an analogy to traffic management, it is more like a set of individually-customizable stop signs and highway patrol officers that define and enforce the rules of the road than it is like a route planner that helps agents find their way to their destinations.

Policies governing authorization, encryption, access control, and resource control are part of KAoS domain management. However, due to our focus on agent systems our scope goes beyond these typical security concerns in significant ways. For example, KAoS pioneered the concept of agent conversation policies (Bradshaw et al., 1997). Teams of agents can be formed, maintained, and disbanded through the process of agent-to-agent communication using an appropriate semantics. In addition to conversation policies, we are developing representations and enforcement mechanisms for mobility policies, domain registration policies, and various forms of obligation policies. These policies are represented in ontologies using the DARPA Agent Markup Language (DAML), and an efficient description logic-based approach is used as the basis for much of the domain manager's reasoning to discover and resolve policy conflicts and to perform other kinds of policy analysis.

The separation of policy specification from policy-enforcement mechanisms allows policies to be dynamically re-configurable, and relatively more flexible, fine-grained, and extensible. Agent developers can build applications whose policies can change without necessarily requiring changes in source code. The rationale for using declarative policies to describe and govern behavior in agent systems includes the following claims: easier recognition of non-normative behavior, policy reuse, operational efficiency, ability to respond to changing conditions, and the possibility of off-line verification.

3.3 Software Agent Domains in CoAX

The CoAX demonstrations contain software agents grouped into agent domains using the CoABS Grid, with the policies enforced by KAoS domain management services. A typical domain configuration is shown in Figure 3.

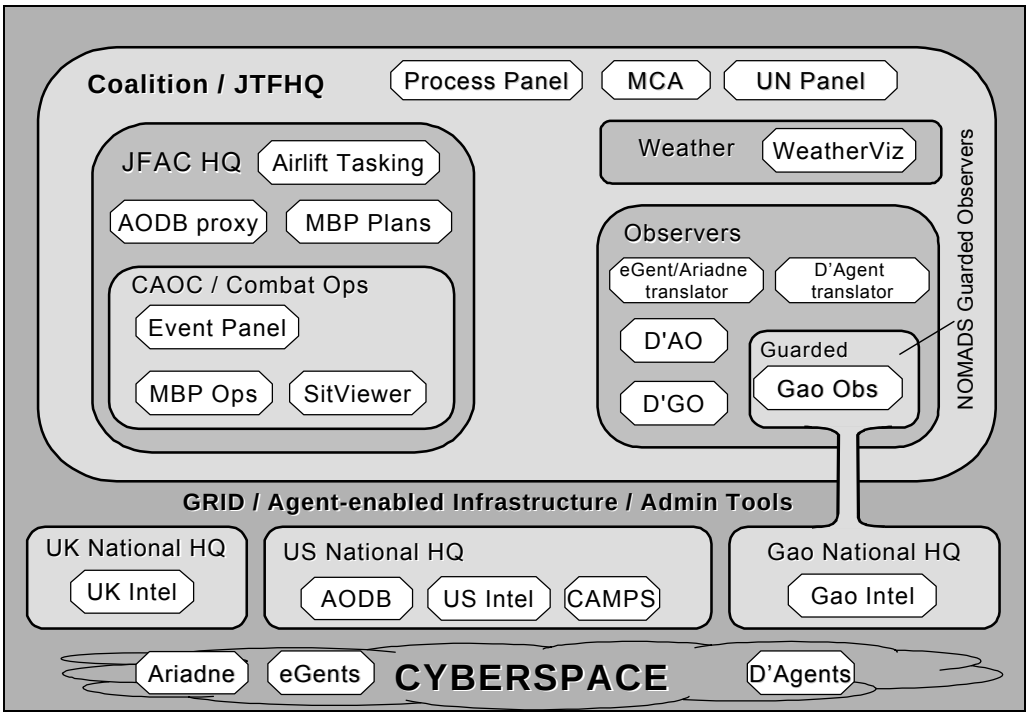


Figure 3. Typical CoAX domain structure; domains are indicated by rounded rectangles; agents by angled rectangles. Some agents are proxies for agents or legacy systems that are not themselves domain aware. Each domain would also contain a Domain Manager agent and a Matchmaker agent (omitted for clarity). Nesting of domains indicates a hierarchy of responsibility and policy control. The agent acronyms are expanded in the body text.

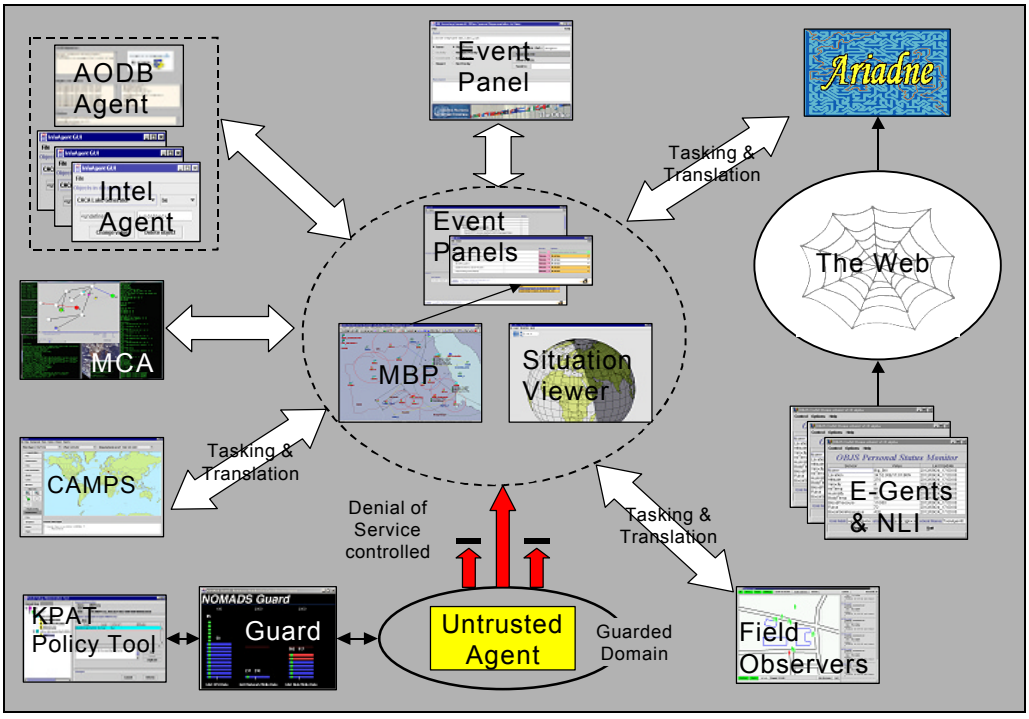


Figure 4: Overview of technologies and agents. The central visualization and planning tools find and acquire data (e.g. disposition of ground forces) and services (e.g. airlift scheduling and plan deconfliction) from the other agents and systems, in some cases via intermediate tasking and translation agents. MBP = Master Battle Planner, MCA = Multi-level Coordination Agent, KPAT = KAoS Policy Admin Tool, AODB = Air Operations Data Base, NLI = Natural Language Interface, CAMPS = Consolidated Air Mobility Planning System.

4 Demonstration Storyboard and Technologies

In this section we progress through the storyboard created for the Binni Scenario, and describe each of the agent systems and technologies brought into play for each part of the story. An overview of the interactions from the agent/system point of view is shown in Figure 4.

4.1 Population of Domains

Following the outbreak of hostilities, the UN has deployed their UN War Avoidance Force for Binni (UNWAFB), to stabilize the region. The active Coalition participants at this time are the UK, US and Gao.

In agent terms, a variety of agent domains are set up using the CoABS Grid infrastructure and the KAoS domain management services, representing the organizational structures (the JTF HQ and the JFAC HQ), the nations (UK, US, Gao) and various functional domains such as Weather and Observers. These domains are populated with a number of agents, which register with their Domain Manager and optionally advertise their services with their domain Matchmaker.

4.2 Data Gathering and Air Planning

After exploring options to separate the opposing forces and restore the peace in the region, the deployment of a large ground observation and peace enforcement force and other courses of action have been rejected, and a 'Firestorm' mission has been decided upon. This will clear land to enable simpler remote and ground observations with less risk to the Coalition peacekeepers. The Coalition undertakes initial information gathering and planning.

4.2.1 Master Battle Planner (MBP)

Air planning at the JFAC is performed using QinetiQ's MBP, a highly effective visual planning tool for air operations. MBP assists planners by providing them with an intuitive visualization on which they can manipulate the air intelligence information, assets, targets and missions, using a map-based graphical user interface (Figure 5). This enables an operator to build a battle scenario containing targets, offensive and defensive units, airspace measures and other objects using simple dialogs and point-and-click techniques on the map. Objects on the map can then be moved around, and their properties can be changed. Information such as the allegiance and status of units, and the ranges of units may also be displayed.

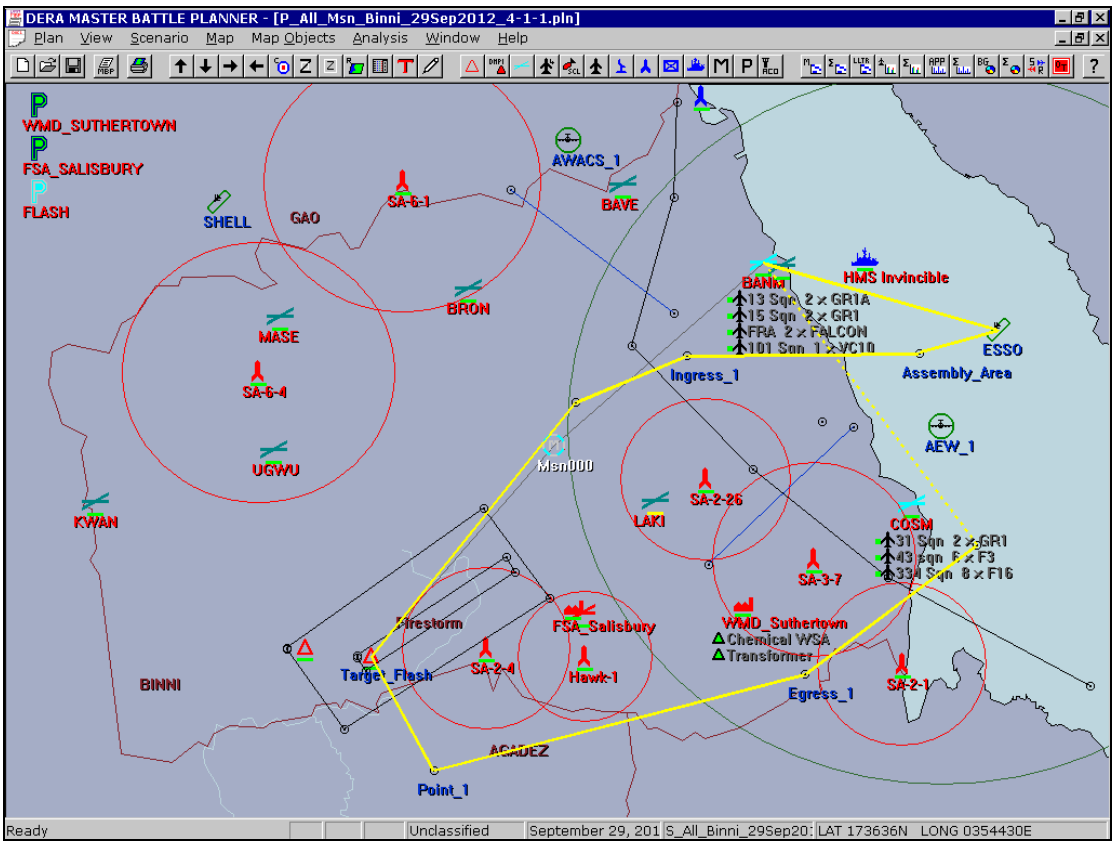


Figure 5: Master Battle Planner map display of the fictional countries of Binni, Gao, and Agadez. A selected mission is highlighted, proceeding from an airbase (BANM), to refueling tanker (ESSO), to the target via waypoints and airspaces, and back to base by a different route.

The operator can interact with these entities and can plan individual air missions (or more complex packages of missions) by dragging and dropping offensive units onto targets on the map. Supporting / defensive elements are added in the same way. The system gives the operator analytical tools to assess the planned air operations for:

- the best utilization of resources; (e.g. by highlighting air units that are over-tasked);
- time-phasing (through charts and animated 'fly-out');
- concordance with the military guidance given.

MBP is a monolithic C++ application, which has been agent-enabled by wrapping it in Java code, using the Java Native Interface. The agent-enabling of MBP allows it to receive all the scenario data (targets, assets, airspaces etc.) from multiple information-providing agents ('Intel Agents' — see Figure 4) and update this information in near-real time. Importantly, this process is integrated into the normal usage of MBP; when an operator views the status of an object, agents are automatically tasked to update the information. Agents may also 'push' changes of status to MBP. Information concerning other air missions can be accepted and merged with missions planned within MBP, as described below. Missions can also be saved and exported, enabling other agents to reason with the data.

4.2.2 Consolidated Air Mobility Planning System (CAMPS)

The second real military system integrated into the demonstration is the Air Force Research Laboratory's CAMPS Mission Planner (Figure 6). CAMPS develops schedules for aircraft to pick up and deliver cargo within specified time windows. It takes into account constraints on aircraft capabilities, port handling capabilities, crew availability and work schedule rules, etc. Users of the planner develop plans (schedules) for aircraft to carry a particular cargo, specifying the intermediate ports, air refueling tracks and the kinds of crews that will be available. They can also specify a number of constraints on the airports potentially involved in the plans to be developed (Emerson & Burstein, 1999; Burstein et al, 2000).

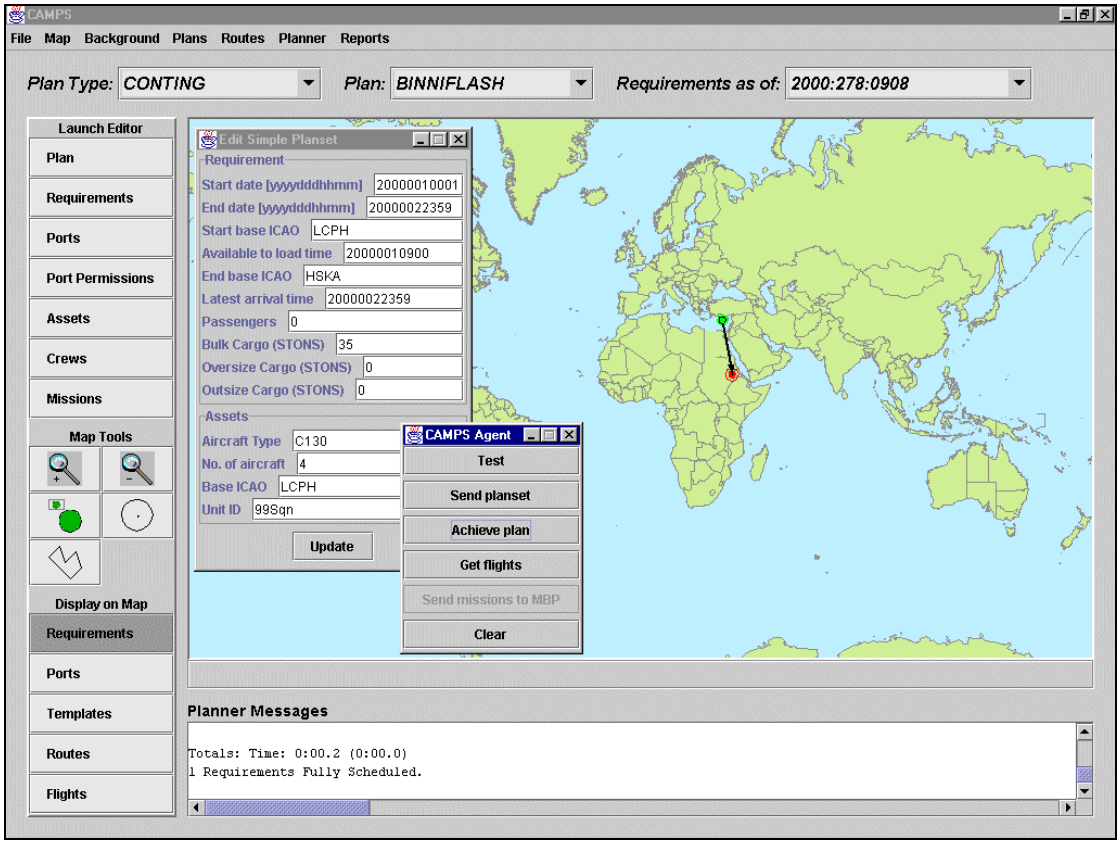


Figure 6: The CAMPS airlift planner, and the demonstration agent used to task the CAMPS agent with a simple requirement: movement of cargo from Cyprus into the fictional country of Binni.

In the demonstration scenario, CAMPS schedules airlifts of cargo into Binni. These airlift flights could conflict with offensive air missions, so the scheduled flights are requested from the CAMPS agent, and sent to MBP, forming part of the normal MBP air visualization. This is achieved by an intermediate agent which tasks CAMPS, and also translates between the KQML messages used by CAMPS and the XML messages used by the MBP agent.

This is an interesting example, as only partial translation is possible; CAMPS and MBP differ fundamentally in their definition of air missions. A CAMPS mission consists of an arbitrary collection of flights, where a flight is a single journey from A to B by a single aircraft. However, an MBP mission consists of a starting point and a route, which must return to the starting point (perhaps by a different path), and may consist of multiple aircraft. CAMPS can therefore produce routes that have no fully valid representation in MBP, although they could be partially represented or indicated graphically.

4.2.3 Ariadne

In a similar manner, weather information extracted from websites by the Ariadne system from the University of Southern California, Information Sciences Institute, is translated and forwarded to MBP, again forming part of the normal picture of the air situation. Ariadne facilitates access to web-based data sources via a wrapper / mediator approach (Knoblock and Minton, 1998). Wrappers that make web sources look like databases can be rapidly constructed; these interpret a request (expressed in SQL or some other structured language) against a web source and return a structured reply. The mediator software answers queries efficiently using these sources as if they formed a single database. Translation of the XML from Ariadne into the XML expected by MBP was handled by custom code, but can now be performed more easily using XSLT (Extensible Stylesheet Language Transformations); this latter technique is used elsewhere in the demonstration (section 4.2.6).

4.2.4 I-X Process Panels (I-P^2)

This Coalition planning process is supported using I-X process panels. I-X is a research program with a number of different aspects intended to create a well-founded approach to allow humans and computer systems to cooperate in the creation or modification of some product such as a plan, design or physical entity — i.e. it supports synthesis tasks. I-X may also be used to support more general collaborative activity. The I-X research draws on earlier work on O-Plan (Tate et al, 1998), <I-N-OVA> (Tate, 1996) and the Enterprise Project (Fraser and Tate, 1995) but seeks to make the framework generic and to clarify terminology, simplify the approach taken, and increase re-usability and applicability of the core ideas. Within CoAX, the I-X approach is being used to provide task and process support and event-response capabilities to various Coalition participants (Figure 7).

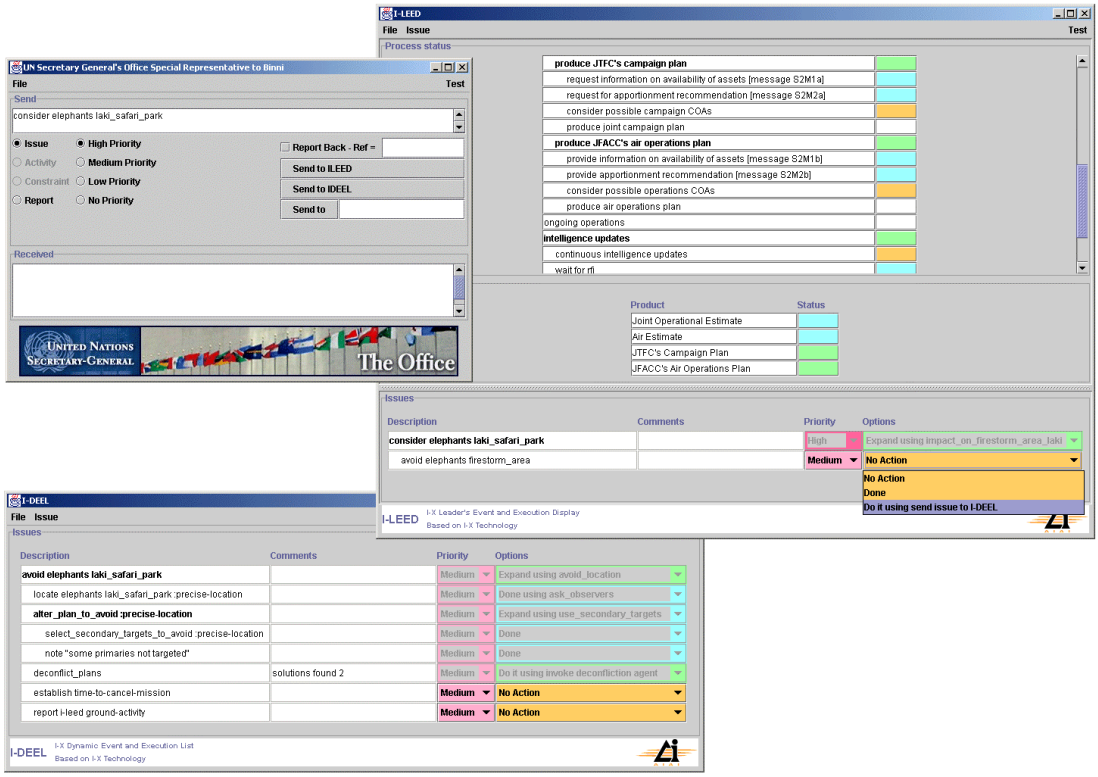


Figure 7: I-X Process and Event Panels

The aim of an I-X Process Panel (I-P^2) is to act as a workflow and messaging 'catch all' for its user. It can act in conjunction with other panels for other users if desired. A panel:

- Can take *any* requirement to:
  - Handle an issue;
  - Perform an activity;

- (in future) Add a constraint.

- Deals with these via:
  - Manual (user) activity;
  - Internal capabilities;
  - External capabilities (invoke or query);
  - Reroute or delegate to other panels or agents (pass);
  - Plan and execute a composite of these capabilities (expand).
- Receives reports and interprets them to:
  - Understand current status of issues, activities and constraints;
  - Understand current world state, especially status of process products;
  - Help the user control the situation.
- Copes with partial knowledge.

4.2.5 Resource control via domain policies

Gao has host nation status within the Coalition but its intentions are unclear and it is distrusted. Special steps are taken to monitor the information passed to and from Gao within the Coalition. During the demonstration, misinformation feeds by Gao (intended to displace the firestorm to allow Gao to take an advantage and move forward) are detected and thwarted. Gao becomes belligerent and launches a denial of service attack against the Coalition's C3I infrastructure.

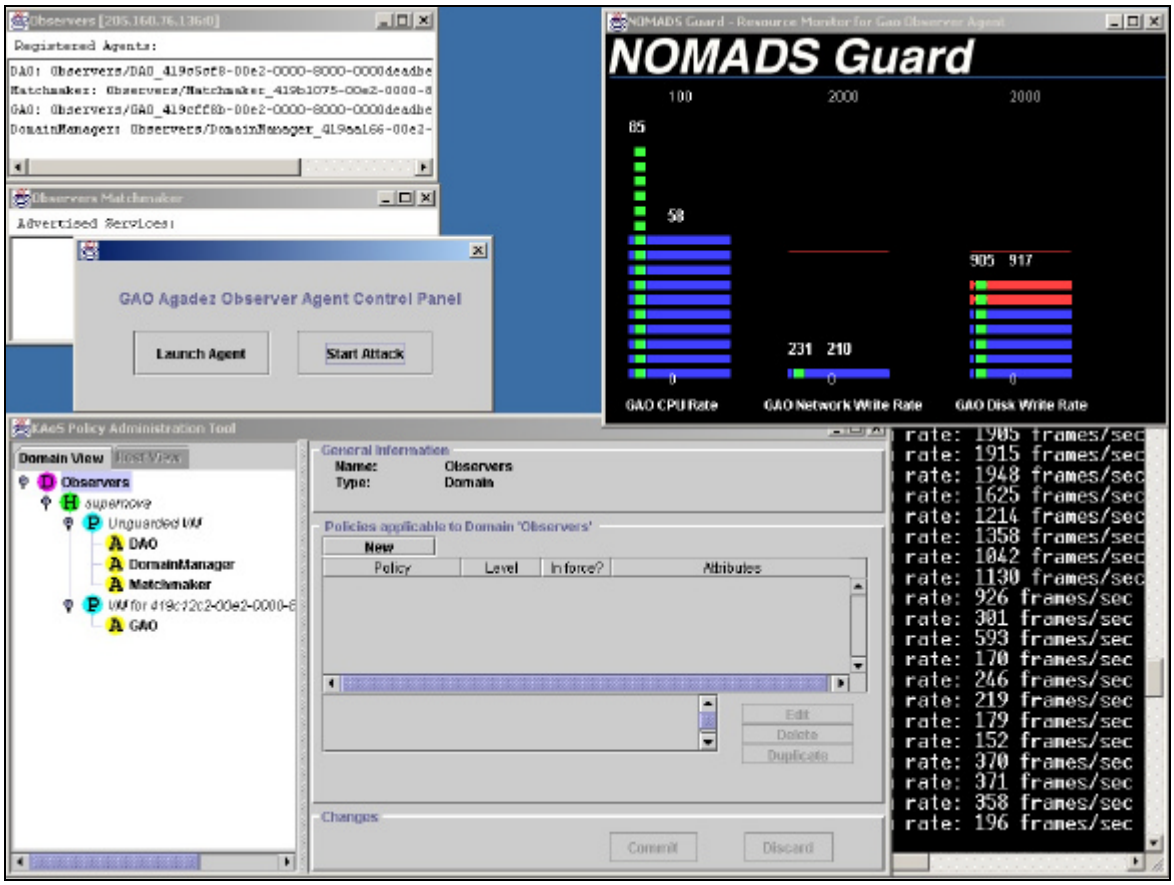


Figure 8: A denial-of-service attack by the Gao agent is starving other agents of resources (note the decreasing rate of processing in the console, bottom right). The Guard (top right) is monitoring the resource usage of the Gao agent. The excessive resource usage triggers a change in domain policy, and the resource limits enforced by the AromaVM are lowered. The policy can also be changed manually using KPAT, the KAoS Policy Administration Tool (bottom left).

The Gao agent in the demonstration is run under NOMADS, a mobile agent system from IHMC. The NOMADS project aims to develop a set of distributed agent-based systems using the Java language and environment. The agent code runs in a new Java Virtual Machine, the AromaVM. The AromaVM provides two key enhancements over standard Java VMs: the ability to capture the execution state of threads and the ability to control resources consumed by threads. By capturing the execution state of threads, the NOMADS agent system provides strong or transparent mobility for agents.

In addition, the resource control mechanisms can be used for controlling and allocating resources used by agents as well as to protect against denial of service attacks by malicious agents. When the Gao agent exceeds certain resource limits, an automatic change in domain policy is triggered by a domain Guard, and the AromaVM is instructed to reduce the resources available to the malicious agent (Figure 8). An operator can manually reduce the limits further, using the KAoS Policy Admin Tool (KPAT).

4.2.6 Data feeds from mobile devices and observers

The firestorm mission has been planned and aircraft have already taken off. However the news media break a story that wildlife in an important safari park in Binni may be in danger as the park overlaps the firestorm area. With only an hour to go, the UN Secretary General's Special Representative to Binni asks the Joint Task Force Commander to consider the wildlife risk aspects of the planned approach. Dynamic information gathering and information feeds using agent technology are employed to create a real time feed of the position of some at-risk large mammals.

This urgent issue is noted and broken down into sub-tasks using the event panels. The progress of aircraft is monitored in near real-time on the Situation Viewer agent from QinetiQ, and the time left before aircraft are committed to their targets is determined from MBP. A search is made for information on the locations of animals in the safari park, and it is discovered that data are available on-line via agents running on monitoring devices attached to large mammals in the park. The agents are eGents (agents that communicate by email) developed by Object Services and Consulting, Inc (OBJS). Historical data from these devices is queried using a Natural Language Interface from OBJS. To aid the planners, a live data-feed is created from the safari park website, using Ariadne to extract data from the pages, and a translator agent using XSLT. The resulting message stream is sent to MBP and to the Situation Viewer agent, allowing the current position and track of the animals to be visualized (Figure 9).

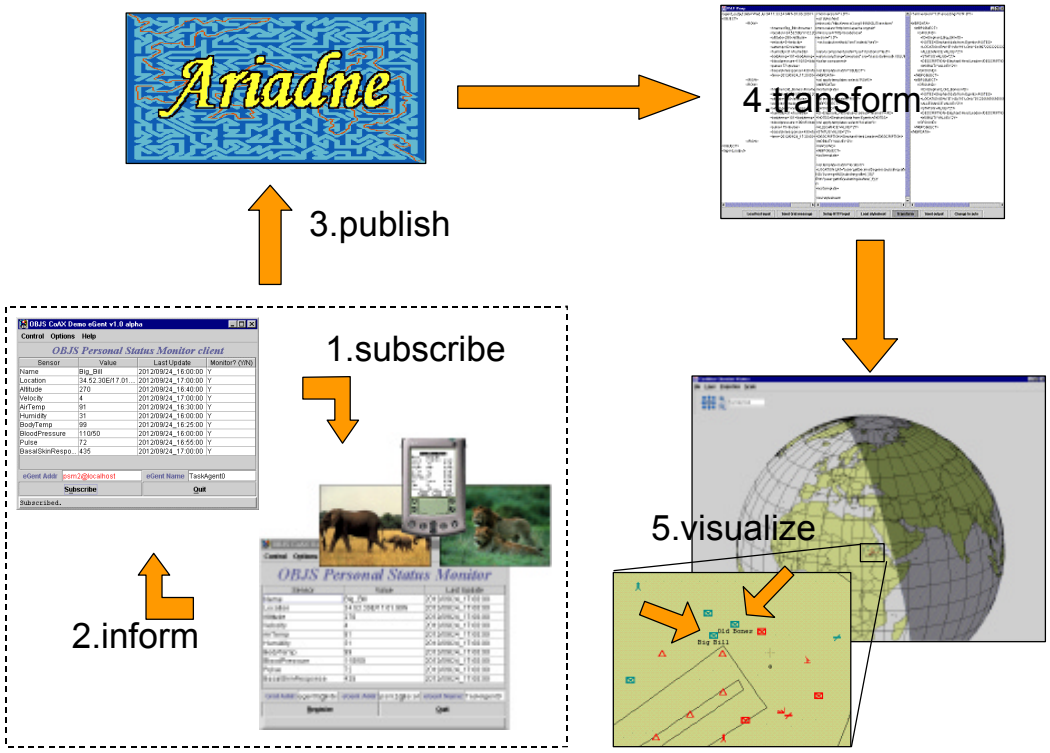


Figure 9: An eGent client subscribes to eGents running on mobile devices (wildlife tags). The data from these devices are published by the client on a web page. Ariadne extracts data from the webpages, and produces XML. The XML is transformed to another format by another agent using XSL Transformations, and finally sent to agents such as MBP and Situation Viewer for visualization.

Data about the movement of ground forces, from the D'Agents field observation system from Dartmouth College, are also transformed using another instance of the translator agent and visualized in the same way, allowing the coalition to identify a convergence of hostile forces on the Laki safari park area.

4.2.7 Plan export and deconfliction

After consideration it is decided to continue with the firestorm mission, but to re-plan as necessary to avoid risk to wildlife. Firestorm targets are adjusted in time or secondary targets selected as necessary for the first wave of firestorm bombing. The impacts of these changes on the Coalition's medical and humanitarian operations are automatically detected, and unintended conflicts between disjoint Coalition operations are avoided.

The air plans are revised using MBP, and then sent to a deconfliction agent to check them against planned activities in other Coalition HQs. The Multi-level Coordination Agent (MCA) from the University of Michigan processes the plans, using multiple levels of abstraction to generate solutions (Clement & Durfee, 1999). The planners are kept informed of progress via their I-X event panels, and can view the results on the MCA display when ready (figure 10). The plans are adjusted iteratively until the conflicts are resolved.

4.2.8 Dynamic Forced Migration (Scram) of Observer Agents

Agadez seeks to use this complication to seize the initiative and launches fighter attacks against a Coalition airborne high value asset (JSTARS) that is monitoring the operation. When this attack is detected, the JSTARS starts to regress, which implies that the observer agents on the JSTARS will not be able to continue providing information to the coalition.

In order to solve this problem, the administrator uses the forced migration (scram) capabilities of the NOMADS mobile agent system to move the observer agents from the JSTARS platform to a secondary ground station platform. The NOMADS system uses the state capture mechanisms in the Aroma VM to capture the full execution state of the agents on the JSTARS. Once captured, the execution state is sent to a new platform where the agents can be restarted without any loss of their ongoing computations (figure 11). This allows the observation agents to continue to operate on the ground station and provide information to the coalition even after the JSTARS regresses.

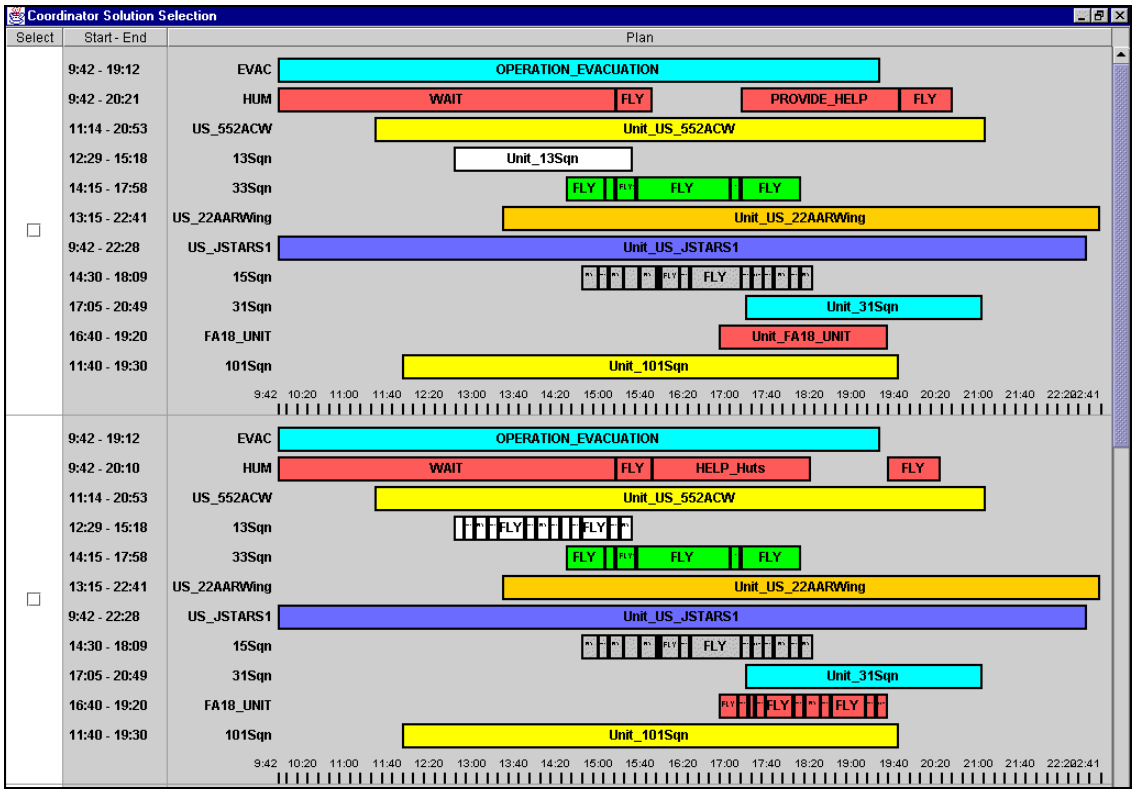


Figure 10: Deconfliction of Coalition plans by the Multi-level Coordination Agent. In the second solution (lower half) two missions (13Sqn and the FA18_UNIT) have been broken down to a lower level of abstraction to seek more optimal coordination

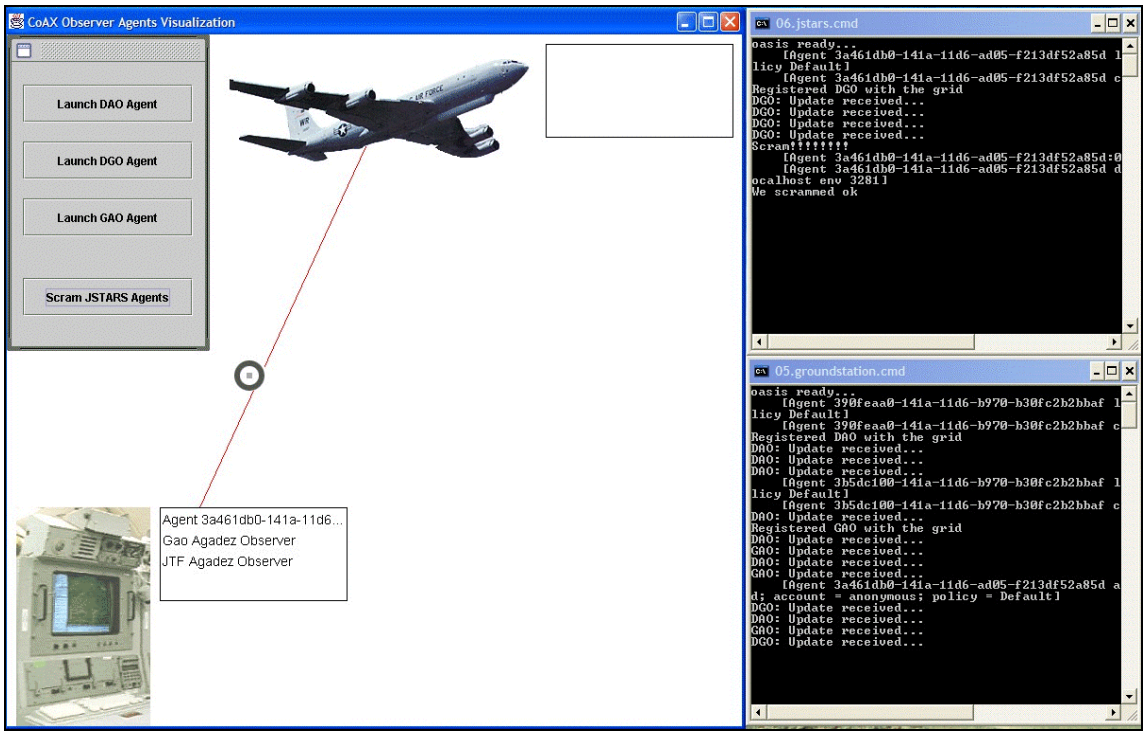


Figure 11: Forced migration of observer agents from mobile platform to ground station, using NOMADS and AromaVM. The updates from the DGO agent, initially on the JSTARS airborne platform (top right console) then start to appear on the new ground platform (lower right console).

5 Assessment of Software Agents

5.1 Technical Progress to Date

The CoAX project officially began in February 2000 and we believe that the demonstrations we have undertaken corroborate the hypotheses outlined in Section 1.3, demonstrating the utility of agent technology in Coalition operations. We have put together a prototype Coalition C2 architecture that supports and embraces heterogeneity and have exercised this in an agent-based C2 demonstration that enacts Coalition activities within the Binni scenario, including both the planning and execution phases of operations.

The CoABS Grid and KAoS domain management capabilities have allowed us to interoperate, for the first time, previously stand-alone US and UK military systems as well as a variety of agent-based information resources. In particular, the CoABS Grid has played a vital role in rapid and robust integration of systems. We have shown how agent organization, behavior, security and resources can be managed by explicit domain policy control.

Assessment work funded by the DARPA CoABS program has reported favorably on the performance issues of agent-enabled infrastructures and the experiences of the CoAX team have shown that the agent-wrapping of legacy systems and the integration of different agent systems at short notice is relatively straightforward. This task is simpler where systems expose more of their internal information and methods. In addition, a heterogeneous set of agents can be made to interoperate as long as implementers adhere to some minimum set of message and other standards. Heterogeneity should be accepted and embraced as it is seen as being inevitable and can actually be beneficial in a number of cases — especially in security terms.

Dynamic task, process and event handling is an important aspect of collaboration and Coalition C2. In the CoAX demonstrations a process panel was used to indicate the start of the tasking and lead into the heart of the demonstration. In the execution phase of operations, process panels in the main commands or headquarters were more extensively used as they enabled a clearer military relevant view of what was happening between the agents in less technical language than would otherwise be visible. Process and event panels have been found to be helpful in keeping users informed of the current stage of collaboration, and maintaining a shared picture of the current state of the collaborative efforts.

Our experience is that an agent-enabled environment gives the ability to create shared understanding and improved visualization. Specific benefits were gained when agents worked semi-autonomously in the background to process

information and support decision making collaboratively with operators, and when agents were integrated into existing tools so as not to disrupt familiar methods of operation.

5.2 Future Research Program

An aim only partially addressed in the current work is the construction and maintenance of a fully dynamic virtual Coalition organization. This would involve:

- domains and agents added to the Coalition structure 'on-the-fly';
- Coalition partners joining / leaving unpredictably;
- handling of dynamic Coalition tasks, processes and events.

Capabilities under investigation for future demonstrations include

- obligation management, e.g. ensure that agents are meeting their commitments;
- improved agent collaboration and run-time interoperability achieved using semantic web languages and technology (Allsopp et al, 2001a);
- richer domain organization and security policies (Bradshaw et. al., 2001);
- richer task, process and event management with more dynamically determined agent relationships (Tate et al., 2002);
- a variety of agents providing new types of data, and data-processing capabilities such as threat classification and track prediction.

Aspects of this work will be included in the Fleet Battle Experiment-Juliet 2002, part of the Millennium Challenge joint integrating experiment.

5.3 Military Implications of the Results

The CoAX research program has shown how software agents can carry out tasks that enable interoperability between information systems and infrastructure services brought together in a 'come-as-you-are' Coalition.

In the experiments so far, the software agents operated in a number of roles. They have worked 'in the background' — through matchmaking, domain management, process management and other agent services — to find, establish and maintain the infrastructure, information and procedural links necessary to achieve and support interoperability in a dynamically changing environment. In addition, they have worked collaboratively with human operators, mediating requests for information and formatting and displaying the results almost transparently.

Thus an agent-enabled environment helps create shared understanding and improves the situational awareness of military commanders. Moreover, it could make a significant contribution to the aims of Network-Centric Warfare which is defined as follows: an approach to the conduct of warfare that derives its power from the effective linking or networking of the warfighting enterprise. It is characterized by the ability of geographically dispersed forces to create a high level of shared battlespace awareness that can be exploited via self-synchronization and other network-centric operations to achieve commander's intent.

One early lesson has been that Cyberspace should not be seen just as an information pipe between humans — it is a Battlespace in its own right. This indicates that 'Cyberspace Superiority' should be obtained (as for any other part of the Battlespace) by ensuring that Coalition forces are able to act decisively through software agents acting on behalf of or mediating the actions of human users.

Dealing effectively with unpredictable changes — owing, for example, to the destructive activities of opponents or because of systems failing and services being withdrawn — is a typical Coalition problem where software agents could make a significant contribution. So far, we have shown that a software agent infrastructure is robust and, to some extent, is 'self-healing'. Our aim is to investigate this further to show that software agents can provide agility, robustness, flexibility and additional functionality beyond that provided by the individual Coalition partners.

6 Concluding Remarks

The central hypothesis being investigated in CoAX is that the agent-based computing paradigm is a good fit to the kind of computational support needed in Coalition operations. The evidence so far confirms this view: we have shown a number of disparate agent systems working together in a realistic Coalition application and indicated the value of the agent-based computing paradigm for rapidly creating such agent organizations. Agents can usefully share, and manage access to, information across a stylized Coalition architecture.

Our conclusion is that software agents, together with agent-based infrastructures and services provided by the CoABS Grid and KAoS, could play a key role in supporting Coalition operations. We think that this technology will provide the ability to bring together and integrate systems quickly to aid in all aspects of Coalition operations, without sacrificing security and control. Our long-term goal is to use this technology in the creation, support and dynamic reconfiguration of virtual organizations — with Coalitions being an archetypal and timely example of an area where this technology is vitally needed.

Acknowledgements

The authors gratefully acknowledge all those who contributed to the CoAX project, including Mark Burstein, Thom Bartold, Maggie Breedy, John Carson, Jeff Cox, Brad Clement, Rob Cranfill, Jeff Dalton, Pete Gerken, Bob Gray, Arne Grimstrup, Paul T. Groth, Greg Hill, Heather Holmback, Renia Jeffers, Martha Kahn, Shri Kulkarni, John Levine, Jean Oh, Pradeep Pappachan, Shahrukh Siddiqui, Jussi Stader, and Andrzej Uszok. The various projects that participated in CoAX were sponsored by the Defense Advanced Research Projects Agency (DARPA) and managed by the U.S. Air Force Research Laboratory, except work by QinetiQ, which was carried out as part of the Technology Group 10 of the UK Ministry of Defence Corporate Research Programme. The US Government and the contributors' organizations are authorized to reproduce and distribute reprints for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA, the US Government, the US Air Force Research Laboratory, the UK MoD, or the contributors' organizations.

References

Alberts, D. S., Garstka, J.J., Hayes, R.E., Signori, D. A. (2001) “Understanding Information-Age Warfare”, CCRP Publication Series, 2001. ISBN 1-893723-04-6

Allsopp, D.N., Beautement, P., Bradshaw, J.M., Carson, J., Kirton, M., Suri, N. and Tate, A. (2001) “Software Agents as Facilitators of Coherent Coalition Operations”, Sixth International Command and Control Research and Technology Symposium, US Naval Academy, Annapolis, Maryland, USA, 19-21 June 2001.

Allsopp, D.N., Beautement, P., Carson, J. and Kirton, M. (2001a) “Toward Semantic Interoperability in Agent-based Coalition Command Systems”, Proceedings of the First International Semantic Web Workshop, July 30-31, 2001, Stanford University, CA, USA, pp 209-228

Bradshaw, J.M., Suri, N., Kahn, M., Sage, P., Weishar, D. and Jeffers, R. (2001) “Terraforming Cyberspace: Toward a Policy-based Grid Infrastructure for Secure, Scalable, and Robust Execution of Java-based Multi-agent Systems”. IEEE Computer, 49-56, July 2001.

Bradshaw, J.M., Dutfield, S., Benoit, P. and Woolley, J.D. (1997) “KAoS: Toward an Industrial-Strength Generic Agent Architecture,” Software Agents, AAAI Press/The MIT Press, Cambridge, Mass., pp. 375-418.

Burstein, M., Ferguson, G. and Allen, J. (2000) “Integrating Agent-Based Mixed-Initiative Control with an Existing Multi-Agent Planning System”, Proceedings of the Fourth International Conference on MultiAgent Systems, Boston, MA, 2000.

Clement, B.J. and Durfee, E.H. (1999) “Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents”, Proceedings of the Third International Conference on Autonomous Agents, pages 252-259, May 1999.

Emerson, T. and Burstein, M. (1999) “Development of a Constraint-based Airlift Scheduler by Program Synthesis from Formal Specifications”, Proceedings of the 1999 Conference on Automated Software Engineering, Orlando, FL, September, 1999.

Fraser, J. and Tate, A. (1995) “The Enterprise Tool Set — An Open Enterprise Architecture”, Proceedings of the Workshop on Intelligent Manufacturing Systems, International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, August 1995.

Jennings, N R. (2001) “An Agent-based Approach for Building Complex Software Systems”. Communications of the ACM. Vol 44, No: 4, 35-41. April 2001.

Knoblock, C. A., and Minton, S. (1998) “The Ariadne Approach to Web-based Information Integration”, IEEE Intelligent Systems , 13(5), September/October 1998.

Rathmell, R.A. (1999) "A Coalition Force Scenario 'Binni — Gateway to the Golden Bowl of Africa'", Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces, (ed. Tate, A.) pp. 115-125, Edinburgh, Scotland, 10th-11th May 1999.

Tate, A. (1996) "The <I-N-OVA> Constraint Model of Plans", Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, (ed. Drabble, B.), pp. 221-228, Edinburgh, UK, May 1996, AAAI Press.

Tate, A., Dalton, J. and Levine, J. (1998) "Generation of Multiple Qualitatively Different Plan Options", Fourth International Conference on AI Planning Systems (AIPS-98), Pittsburgh, PA, USA, June 1998.

Tate, A., Dalton, J., and Stader, J. (2002) "I-P^2 — Intelligent Process Panels to Support Coalition Operations", in Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations (KSCO-2002) (ed. Tate, A.), 23/24-Sep-2002, Toulouse, France.